1-7-2014

Divy: A Website For Facilitating The Sharing And Purchasing Of Digital Content

Aidan Crosbie
Santa Clara University

Lauren Falzarano
Santa Clara University

Nicole Pal
Santa Clara University

Santa Clara University

DEPARTMENT of COMPUTER ENGINEERING

Date: January 07, 2014

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Lauren Falzarano, Aidan Crosbie, and Nicole Pal

ENTITLED

FileHub: A Website for Facilitating the Purchasing and Sharing of Digital Content

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREES OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

or

BACHELOR OF SCIENCE IN WEB DESIGN AND ENGINEERING

THESIS ADVISOR

DEPARTMENT CHAIR

Divy: A WEBSITE FOR FACILITATING THE SHARING AND PURCHASING OF DIGITAL CONTENT

by

Aidan Crosbie, Lauren Falzarano, and Nicole Pal

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
or
Bachelor of Science in Web Design and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
January 07, 2014

1 Abstract

For our senior design project, we designed and built software that allows people to easily sell and share digital content with others. Our system was implemented as a website where users may upload any kind of digital content of their own creation, and specify the price for which other users can download it. A user can upload up to three gigabytes of data for free; however, if they wish to charge money for downloads, our site will take a percentage of the download's/downloads sale price. A number of technologies were used in order to accomplish this, including PHP, HTML5, CSS, JavaScript, CoffeeScript, and a graph database. Design and logical specifications remain incomplete as the team continues to work out the logic behind it. The end goal is to create a place where content can be shared and a community can be formed around said content as well.

2 Acknowledgments

We would like to thank our advisor, Daniel Lewis, from the Computer Engineering department. We'd also like to thank our english professor, Jackie Hendricks, as well as all other the mentors and teachers who have provided us with the necessary knowledge to complete this project. Finally, we would like to thank our friends and families who have supported us through out the process.

Contents

List of Figures

List of Tables

3 Introduction

3.1 Problem Statement

Currently, there are many websites that offer a way for users to upload various kinds of digital content, including music, video, images, et cetera. However, the majority of these specialize in only one type of content. If a user wishes to upload any combination of the above stated content types, there is no site that allows them to do so. In addition, most existing sites function only to showcase said content. The majority do not have a way for the uploader to sell their content, or do not have a way for the uploader to sell it easily. Additionally, users often do not have the option to customize the look and feel of the pages where their content is.

We built a new system: a website that will allow users to upload any kind of digital content of their own creation, and specify the price for which it can be downloaded. To ensure that our site is capable of sustaining itself financially, a portion of the money paid for downloads will come back to us. Purchases will be processed through our site, using PayPal. Additionally, users will also be able to customize their profile pages, determining the graphics and color scheme, what content is displayed where, and so on. They will also be able to join with other users to make group pages, as members of a band might want to have separate pages for themselves, and a joint one for their band. In addition, there will be a social network aspect to the site users will be able to see updates from their favorite artists, post content on their own pages, connect with others, and so forth.

3.2 Background and Motvation

Currently, there are a number of different websites and systems that people may use to exchange, download, listen to and purchase digital content. None of these, however, offer all the features we plan to implement. We have listed a few similar systems below and explained how they differ.

- Bandcamp [1] is a website that serves as a platform for musicians and labels to sell their music. It provides them a great deal of control – they can specify the price of each file and album, customize the look of their profile pages, include art and video with music purchases, view statistics, sell music on Facebook, offer discount codes, sell physical copies of albums, and offer pre-orders. For an additional cost, sellers can integrate with Google Analytics, offer more than two hundred downloads a month, and acquire a custom domain. It provides a lot of features, but differs from our proposed project in that it only allows users to sell music. It is possible to offer additional art and video, but these have to be bundled with music, and cannot be sold separately.

- iTunes [2] is a desktop application that allows users to search for and purchase audio, music videos, television shows, and movies. However, it is limited in that users cannot upload their own work – the only content in the iTunes store comes from artists signed by record labels. It is also limited in that the only types of content a user may download are the ones listed above.

- SoundCloud [3] is a website that allows users to upload their music to a unique URL, and then share and embed it in other websites. Other users can then search for, listen to, and embed the music on this site. However, SoundCloud is limited in that it does not allow users to sell or purchase content, and does not support any content that is not music.

- Spotify [4] is a desktop application that can be used to stream and listen to music. However, it only offers music from signed artists, and does not allow users to upload their own content or purchase other music. In addition, there is no social aspect, and the artists themselves complain that Spotify does not provide them with adequate profits.

- YouTube [5] is a website that allows users to upload videos that can then be watched by other users. Apart from the fact that it only supports one type of digital content, YouTube is limited in that it does not let users upload high quality videos, and that users cannot download videos. It does, however, have something of a social aspect; users all have profile pages that they can customize, and they can interact with one another by sending messages and commenting on other users' profiles.

3.2.1 Summary Table

Features	BandCamp	iTunes	SoundCloud	Spotify	YouTube
All Digital File Types Supported					
All Users can Upload Files	X		X		X
Social Networking Elements	X			X	X
Users can Purchase Files	X	X			
Users can Sell Files	X		X		
Users have Profile Pages	X	X	X	X	X
Users have Highly Customizable Profile Pages	X				X

3.3 Objectives

Our main objective, of course, was to build a functional website. We wanted this website to be the best we could make it, and thus, we kept the following in mind as we planned, built, and refined our project.

- Phase One: Planning
  - Determine functional and non-functional requirements for the system, so that we know everything it's got to be capable of.
  - Determine the different components of the system and how they will interact with each other.
  - Determine what languages and frameworks are best to use.
  - Design database.
  - Make mock-ups of the different pages.
- Phase Two: Implementation
  - Implement database. This will involve creating a SQL database that each team member can access, setting up the tables according to the design from the planning stage, and populating it with practice data.
  - Implement front end of the website.
  - Add basic functionality to website: create account, upload/manage content, etc.
  - Add payment system using PayPal API.
  - Add security features
  - Perfect the design of the site.
- Phase Three: Testing and Revising
  - Test functional features to see if errors can be found.
  - Do extensive vulnerability testing to look for weaknesses.
  - Make any appropriate changes

4 Requirements

We created a set of functional and non-functional requirements of the site to further establishe the parameters of our project. Functional requirements list what the user will be able to accomplish and the behavior of the site. Non-Functional requirements determine the accesability, security, and overall performance of the site.

4.1 Functional Requirements

- User can upload digital content and set a price
  - Users will be able to upload various forms of digital content that they wish to publicize
  - Content can be sold for a price or set for a free preview
  - Content can also be removed
  - A user can adjust the download price of their content at their disgression
- Artists must have profiles
  - Give users the ability to highlight content they wish for other users to view or purchase
  - Will allow users to subscribe to other artists
- Profile required to purchase content
  - Users will be required to be a registered member of the page to purchase content
  - If content is free, a profile is not required to view content
- Users can create, edit, and delete accounts
  - Users can monitor their activies and interactions through their account
- Users can personalize profiles
  - Users have the ability to upload a profile picture
  - Users have control over the location of content on their profiles
- User may rate uploaded content
  - Users will be able to rate digital content through a five star rating system
  - All users may view content's rating as it will appear underneath it
- Tag content so that it easily searchable
  - Users will be able to search for content based on digital content format
  - Users will also be able to search for other users and artists
- Terms of Service
  - Users must agree that there will be a small royalty fee on all transactions to continue hosting services

  - Users will also agree that digital content is their own original work and that they will not freely share other users' content

- Responsive Web pages
  - Users must be able to easily and enjoyably use Divy on any sized window and the it will adjust as they change the size of the window

- Mobile ready Site
  - Divy must be accessible from all internet capable devices,with the scaling dependent on the screen size

4.2 Non-Functional Requirements

- Usable by non-technical users
  - Users must be able to navigate the website easily
  - Users must be able to upload and download content with ease

- Safe for minors to use
  - Content must be appropriate for all audiences
  - Content that is graphic must be able to be removed and reported

- Maintainable by an administrator
  - An administrator will be able to edit pages if needed
  - An adminstrator will be able to remove content that goes against the terms of agreement

- Aesthetically pleasing and modern looking
  - Colors must not be too contrasting
  - Any video or graphic must render correctly on the page

4.3 Use-Cases

This site provides users with a multitude of features. As such, providing a use case for each and every single function is beyond the scope of this document, but we have provided the details for the most important and common cases.

4.3.1 Use Case 1

Actor: New User

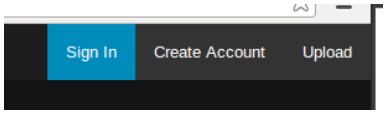
Goal: Create an account

Precondition: The user must have a computer with internet access, and an email account.

Postcondition: The user has an account on our site.

Scenario:

1. The user goes to our website.
2. The user clicks the 'Create Account' button on the navigation bar, which will direct them to the registration page.



3. The user fills out all of the required fields on said page, which involve creating a username, entering a password, specifying an email address, and entering security questions in the event that the password is forgotten.

4. The user presses a 'Submit' button.
5. The user checks their email for a message containing an account validation link.
6. The user clicks on the link to validate their account.

Exceptions:

1. There is an error while filling out the fields on the registration page. For example, a user could enter an incomplete email address, a password that does not meet the requirements, a username that is already taken, and so on.
   (a) Error messages will appear on the page as the user is typing, informing them of any errors in need of correction.

4.3.2 Use Case 2

Actor: User

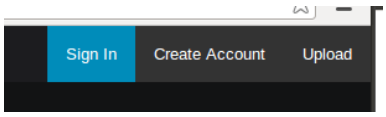
Goal: Upload content

Precondition: The user should be on the website and logged in to their account.

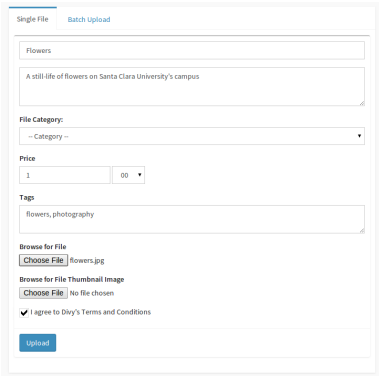
Postcondition: The user's content is in our system and available for download.

Scenario:

1. The user clicks the 'Upload Content' button on the navigation bar, which will direct the user to the upload page.



2. The user fills out all of the fields on the upload page, which would include specifying a title for the content, a short description, keywords, a price, the type of content, and so on.



3. The user clicks the 'Browse Computer" button to specify the file from their computer that will be uploaded.
4. The user reads through the User Agreement, and clicks a checkbox saying that they have read and agree to said Agreement.
5. The user presses the 'Upload' button at the bottom of the form.

Exceptions:

1. One or more of the fields has not been filled out by the user.
   (a) Error messages will appear on the page after the user presses the 'Upload' button, with instructions on how to rectify the errors.
2. An error occurred while attempting to add the content to the site database.
   (a) The user is notified that an error occurred and given the chance to attempt the upload again.

4.3.3 Use Case 3

Actor: User

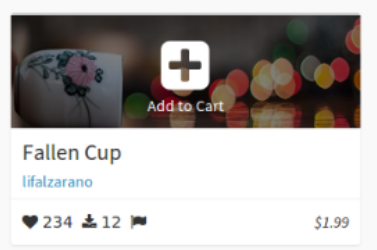
Goal: Purchase content

Precondition: The user needs to be on the website and logged in to their account.

Postcondition: The content chosen by the user has been downloaded to their device, the user's account has been charged, and the artists have received their payment.

Scenario:

1. The user navigates through the site, looking for content they want.



2. The user finds content they want and presses "Add to cart".
3. The user decides that they have added enough content to their cart, and presses the "Checkout" link.
4. The user views their cart, and decides if they want to check out or not. If they do, they press the "Proceed to Checkout" button.
5. The user enters their payment information, and presses 'Finish Purchase.'
6. The content downloads to the user's computer.

Exceptions:

1. One or more of the fields has not been filled out by the user during checkout.
   (a) Error messages will appear on the page after the user clicks away from the field being filled out, with instructions on how to rectify the errors.
2. The user's payment information is found to be invalid.
   (a) The user is notified that an error occurred and given the chance to specify their information again.

4.3.4 Use Case 4

Actors: User with uploaded content, a third party that wishes to flag said user's content, an employee of our site who will review the claim.

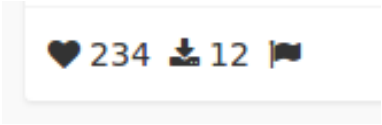
Goal: The third party wishes to flag specific content, and possibly get the content removed from the site.

Precondition: The content in question is already on the site, and the third party needs to have an account.

Postcondition: If the content is found to violate any American laws or site policies, it will be removed, and the offending user's account potentially deleted. If it is explicit content that was not marked as such, it will be marked. If it is not found to violate any laws or policies, no action will be taken.

Scenario:

1. The third party finds content that they decide to flag, either because it they believe it's not the property of the user who uploaded it, they believe it contains explicit content, they feel that it contains inappropriate or hateful content, or for any other reason that content can be flagged.
2. Said third party presses the "flag" button located at the bottom right corner of the content.



3. In the pop-up box that appears, the third party specifies why the content is being flagged.
4. The third party presses the "submit" button in the pop-up box.
5. The content will be marked as "flagged for removal" or no longer listed until it has been reviewed by a site employee depending on the reason provided for it being flagged
6. The employee of our site receives the claim, reviews it, and decides on the proper course of action.
7. If the content is removed, it will still be listed for a month after removal to let users know what happened, but it will no longer be viewable or downloadable.

5 Design

5.1 Design Rationale

Before we settled on creating a website, we considered making a local desktop application. Several existing products have been implemented as such - e.g. Spotify and iTunes - and we thought it would make sense to do our project in a similar fashion. In addition, some of us have experience making GUIs (graphical user interfaces), and like doing so.

However, as we looked into it further, we realized that a local application does not have any advantages over a website, but would have substantially more drawbacks. Some of these are as follows:

- We would have to deal with cross operating system compatibility - that is, we would have to make sure that it works on different versions of Windows, MacOS, and Linux.
- We would not be saved from having to deal with cross-browser compatibility, because we would still need a website from which users could download the application in the first place.
- It would be a lot harder to push out changes to the product. We would need code that would install updates each time we make substantial updates to our system.
- We also run the risk that a user will not install new versions of our software, which means that we might not be able to add new features that are not compatible with previous versions.
- A desktop application is hardly more secure than a website. We would still need a user account system that would be stored in a database on our servers, as we would like users to be able to log in to our software from any computer. We would also still need to send payment information over the internet.
- It is much harder to customize the look and feel of a desktop application. Nobody on our team has extensive experience with native application graphics, so it would be difficult for us to tailor a GUI to look exactly as we want it.

In the end, we decided that making a website would be best. We found that doing so has a number of advantages, listed below:

- All of us are very familiar with making websites. Between the three of us, we have extensive experience with front-end and back-end development.
- It is far easier to customize the look and behavior of a website.
- Users do not have to download any software in order to use our product; they can simply visit the site and start browsing the available content. Thus, we are more likely to get customers if we make a website.
- A website also allows us to make use of existing plugins to showcase certain kinds of content. For example, there are a lot of existing music players and photo albums out there, so we wouldn't have to code such things from scratch.

5.2 Technologies Used

- PHP [6] is a scripting language that can build dynamic webpages. PHP is easily compatible with HTML5 and can allow for pages to be generated based on the user logged in and what artist page they are currently viewing. It allows us to easily connect to the backend server where the digital content is stored while keeping the content secure. Scripts in PHP may be re-used on different pages, reducing effort times drastically.

- HTML5 [7] is becoming the new standard for webpages in the near future. It allows for syntax that is simpler to read so that pages can be built much faster. It also supports audio and video, which is a must for a digital content website such as this. HTML5 reduces cross-browser problems and is compatible with most modern browsers, such as Chrome and IE9, so the need for different code will be minimized. It also is easily adapted into mobile browsers which allows for users to access the website even if they aren't near a desktop or laptop. Its local storage system allows for easier caching of user information.

- CSS [8] gives webpages a certain style and feel. It makes it simple to maintain a uniform look and feel throughout all pages on the site. CSS has also been used to adapt the site to older browsers such as IE6 so that pages will be rendered correctly.

- JavaScript [9] can be used to provide client-slide functionality and dynamically change pages after they have been loaded in PHP. When a user fills out a form or registers for the site, JavaScript saves time by showing an error without forcing the user to completely restart filling in the form.

- JQuery [10] is a feature rich JavaScript library that adds to the dynamics of a website.

- A Neo4j [11] graph database stores user information and digital content in a secure and efficient way. In coordination with PHP, information can be encrypted so that user information cannot be leaked. This is of utmost importance because if the information is leaked, a third party member may change critical information, such as where to send money after the purchase of digital content.

- The PayPal API [12] is for purchasing and selling content on the website. This will be an easy and convenient way for users to sell and buy content. This is an established and secure method of transactions and will remove any complications that will arise in the use of credit card transactions.

- Bootstrap [13] Bootstrap is a front-end framework that provides basic styles to build our site. The open source code allowed us to customize it further while providing us with a basic style/basic styles that had high levels of polish which allowed us to quickly iterate through large design changes

5.3 Site Design

The system is implemented as a website containing a number of different pages, each serving a different purpose:

- Home Page: This page is the central point of our site. It contains sections that display the most popular content of the site, with different sections for different file types. This content is tailored to match the interests of the user when such data is available. If the user is logged in, it will act as a kind of feed relating to their history. If the user is following any artists their updates are shown here.

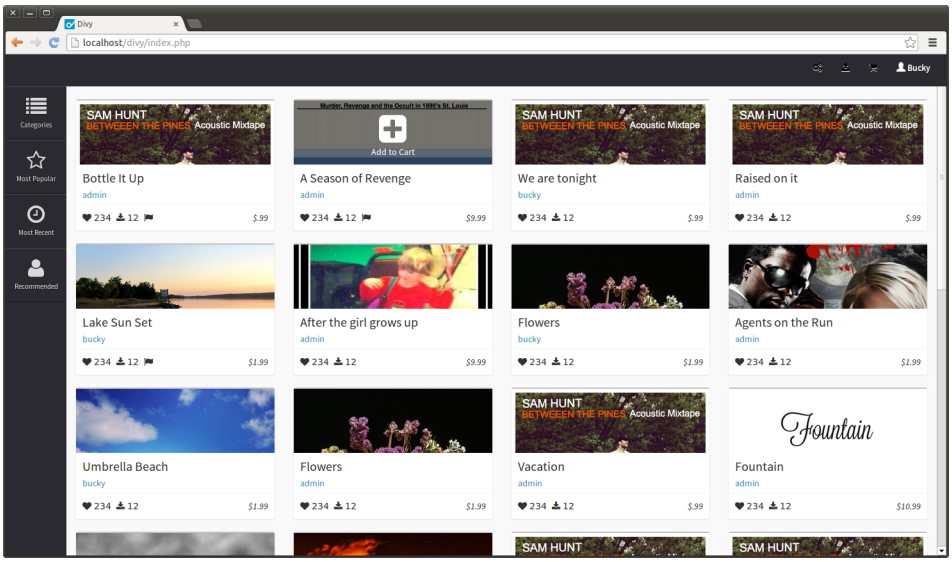


Figure 1: Home Screen

- Profile Page: This page contains a variety of things, and can be completely customized by the user. They have the option to display files they have uploaded that are available for download, and we provide multiple ways of displaying it nicely. For example, we have music players that they can choose from to display music, photo albums for photographs, pop-up document viewers for PDF's, and so on. We also give them the option to have a section that will contain status updates and public messages from other users.

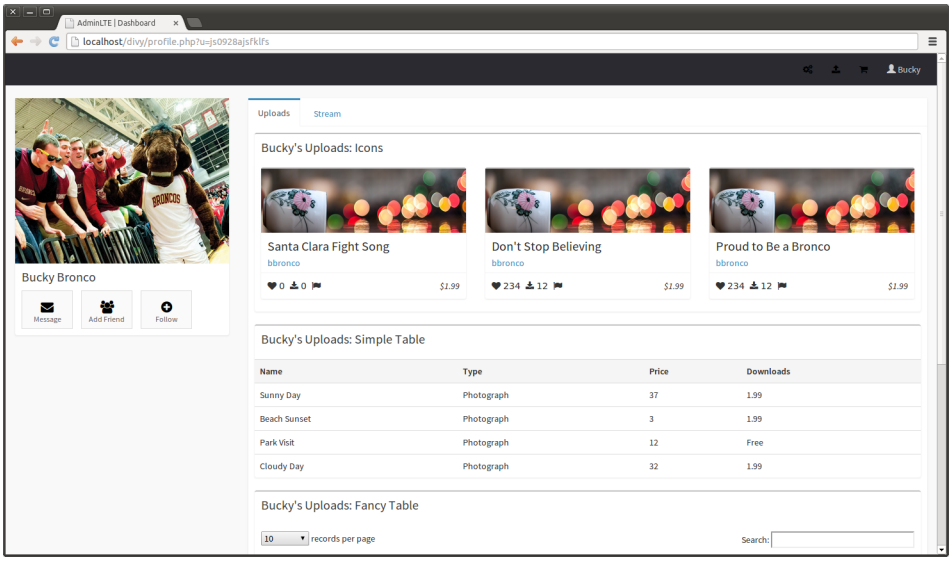


Figure 2: Profile Page

- Graph Database: This is used to store all the site's data, such as account systems, locations of digital content, purchase history, and so on. It is also the basis for the recomendation system used on Divy. As users update or change information through the web interface, our PHP backend updates the database accordingly. In addition, the database is queried each time a page is loaded, as information on the most popular content is stored there, as is information on users' preferences for the appearance of their profile pages. Specifically we have chosen the Neo4j graph database to store our data. The graph database and cypher querries allos for the optimal balance between data storage and data traversing for each querry which is why we chose to use Neo4j.

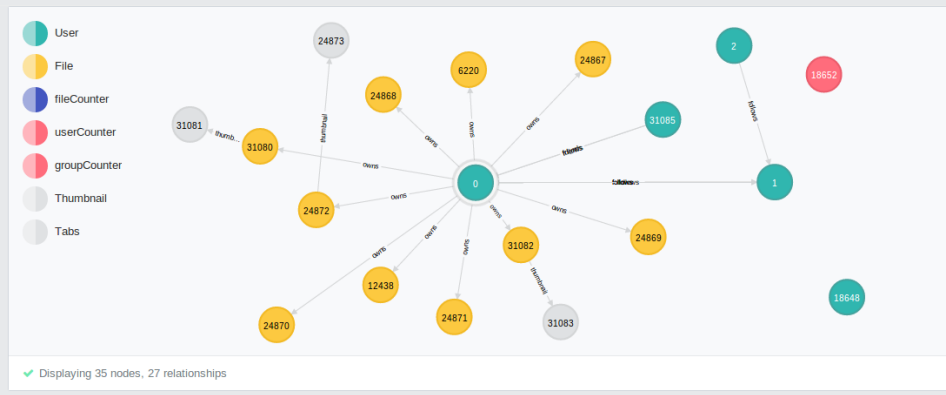


Figure 3: Graph Database

- Web Server: We're using the Google Cloud [17] platform to host our website, database, and all content. Users access our site through Google.
- Recommendation System: Divy will provide recommended content to logged in users who have viewed or liked files. This information, coupled with other user's file views and likes, allows for relationships between users and files to be quickly analyzed. As the user begins to view and like more content on Divy, more accurate recommendations will be made upon a weighted universal user rating of the file, total file views, and the users prior viewed and liked files.

6 Conceptual Model

To help design the website, the following conecptual models have been created to show interconnections between the user and database, how the website will function, and a paper prototype of the user interface.

6.1 Data Flow

The data flow model in the figure below displays various ways that users complete tasks by pushing information through the website. These would include putting content up for purchasing, creating a new account, and purchasing content.

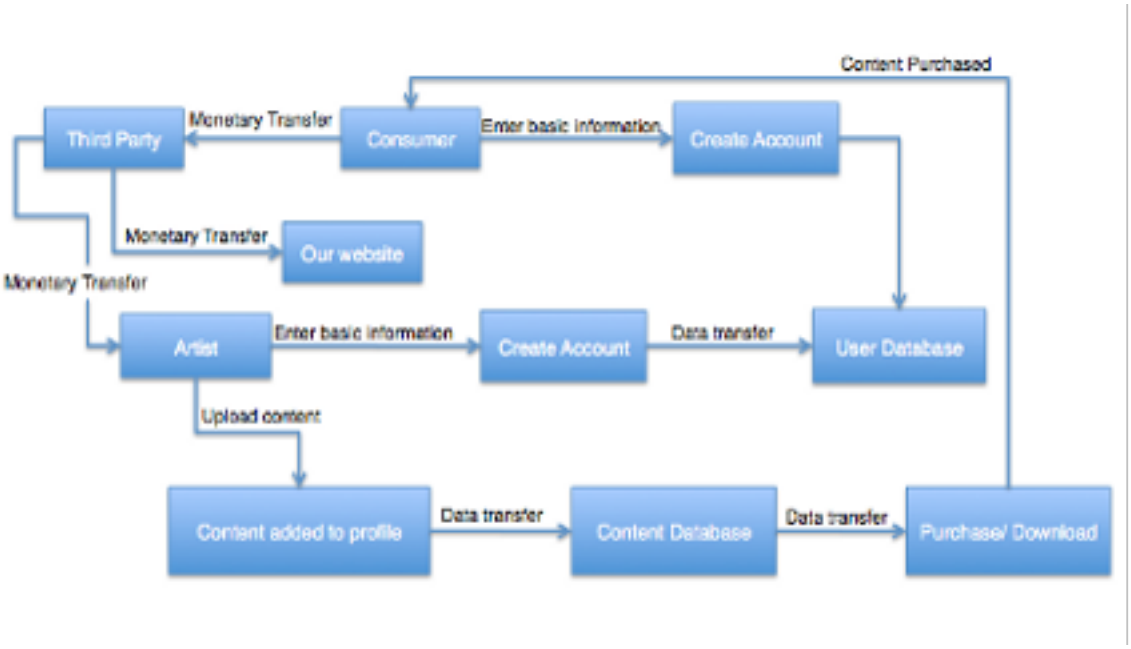


Figure 4: Data Flow Chart

7 Testing and Verification Plan

In a large scale project such as this, we must test the frameworks so that automated tests can be run. Our functions need to be easily tested on a regular basis, and by following the testing plan below our development team has the ability to ensure the website is functioning correctly.

7.1 Unit Testing

- Frameworks for unit testing includes:
  - PHPUnit [14] - a programer oriented testing framework designed for PHP unit testing.
  - JSUnit [15] - a unit testing framework for JavaScript.
  - Selenium [16] - used for automating web applications for testing purposes.
- Each framework provides a way to run automated tests on code segments
- Developers run unit tests on code they write
- Black box testing is only concerned with the functionality without peering into the internal structure of the software, and White box testing examines the internal workings as opposed to the functionality of the software. Both may be performed by the developer
- Testing lead re-runs unit tests to ensure code correctness
- Tests are run on a biweekly basis

7.2 Alpha Testing

- Testing lead developed tests to ensure the website's interface and functionality is working properly
- After changes are made to the website, testing lead retests the website
- These tests occur on a weekly basis, unless changes have been made to the website
- Routine tests are documented

7.3 Acceptance Testing

- When the website has functionality, typical users or beta testers are invited to begin using the site
- They were asked to use the website normally and provide recommendations on how the site could function better
- These recommendations may be implemented if they allow for better usability of the site

7.4 Test Cases

The following are a list of several of the test cases used throughout the development of the application. Each includes a priority level, the steps in conducting the test, and what conditions must be met for a passed test. Priority levels are labeled as is follows: 1 for critical, 2 for convenient with a workaround available, and 3 being for a cosmetic problem that will not affect functionality of the application.

Priority	1
Scenario	1. User logs in 2. User clicks upload content and selects desired file to upload
Condition for Passed Test	Content is seen on the website with the specified price, and is only downloadble with certain permissions

Table 1: Test Case for Uploading Content

Priority	1
Scenario	1. User fills in registration 2. User accepts terms of agreement
Condition for Passed Test	User receives an email with their login information

Table 2: Test Case for Registering

Priority	1
Scenario	1. User selects content they wish to purchase 2. User fills in payment information. 3. User agrees to not illegally share content by accepting terms of agreement
Condition for Passed Test	User is redirected to a new page and begins downloading content

Table 3: Test Case for Purchasing Content

Priority	1
Scenario	1. User uploads content or registers for site 2. User clicks submit
Condition for Passed Test	User Information is stored in the database and the database is safe from outside attacks

Table 4: Test Case for data being sent to database

8 Societal Issues

As we were building our project, we were not focusing solely on making a high quality product, but were also considering the implications our work could have on others. The very nature of our project - a website for file-sharing - brings up a great deal of issues, and we have gone over our approach for dealing with many of them below.

8.1 Ethics

Our project brings up the following ethical issues: intelectual property, hate speech, explicit content, and child pornography. In this section we have outlined each of them in turn.

8.1.1 Intellectual Property

First and foremost, we must deal with the ethics surrounding ownership of property. Our site would enable anybody to upload absolutely any file that they have got in their possession. Of course, it's quite simple to acquire files that are another person's intellectual property these days, and we can be absolutely certain that at least one user will upload content that is not theirs to the site. This, of course, is ethically wrong; if the original author of the content is selling said content elsewhere, they are being robbed of their profits. If the person who stole it and uploaded it is charging money for said content, the original authors are being robbed further still of their due profits. Thus, we need to have some sort of protection.

It does not make sense for us to screen all content before it goes public to make sure that it does not belong to somebody other than the party uploading it. This is simply way too difficult; it is not feasible to check one single file against each and every copyrighted file in existence, let alone check every file that is uploaded to our system. There are simply far too many copyrighted files out there – we do not have the means to check them all. Instead, we were planning on implementing a system similar to the one that is used by YouTube and many other content-sharing sites: when a user submits a file to our system, they have to check a box saying that they agree to our user agreement. Said agreement will contain speech that states that this file genuinely does belong to the party uploading it, and that they do have the rights to distribute and make a profit from it (should they wish to charge). If another party encounters this content on our site, they would be able to flag it and claim that it belongs to them.

We would need to consult with a lawyer before determining the next steps – if our site ever is available for public use – but we imagine they would suggest the following. If we trust the party making the dispute – that is, if they actually are a company or artist that is known to possess intellectual property. We would temporarily block the content from being accessed on our site. If either of the two parties could present us with proof of copyright, we would either unblock the content or outright remove it, depending on who the actual owner is. If either party wishes to sue the other, that is between them, and we should not be involved at all. In any case, we would protect people who possess intellectual property with such a system.

8.1.2 Hate Speech

Assuming that the files a user uploads are their own intellectual property, and that they're not malicious, there is still the possibility that they contain hate speech or are explicit. We want to completely prevent the former; unlike certain other websites that have notoriously lax policies on

hate speech, we do not intend on tolerating anything that contains such. However, it is not practical for us to manually screen each and every file uploaded to our system, and automated tests can only catch so much. Thus, users will have the option to flag a file on our site, and specify that it contains hate speech. We'd have to have a human employee look over all flagged files, and determine if they should be removed.

Secondry hate speech could also appear in the comment sections on the site. Once again, we will have a no tolerance policy, and if comments are reported and verified to contain hate speech, the account will be shut down and banned from returing to Divy.

8.1.3 Explicit Content

We also need to be concerned about explicit content. We do not want to completely prevent such content from being uploaded to our site, we just want to make sure that users are aware ahead of time that a file is explicit. Thus, when uploading content, users will be presented with a 'mark as explicit' box that they can uncheck. By initially funneling all content into explicit content, it will require uploaders to consiously make the decision as to whether their content is safe for minors. All explicit content will be clearly labeled as such, and if a user is under eighteen years of age, they will be barred from viewing and downloading. If users come across explicit content that is not marked as such, they can flag it. Again, a human employee would have to verify this claim.

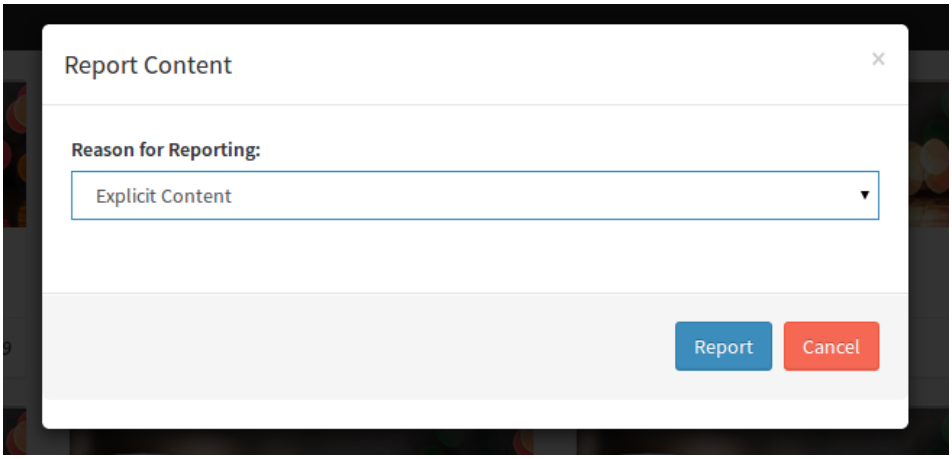


Figure 5: Reporting Explicit Content

8.1.4 Child Pornography

We have also considered the possibility of child pornography being uploaded to our site. We would have a zero tolerance policy for this – anybody caught uploading or knowingly downloading such files would be banned from the site for life. We would also want to actively check all content marked as explicit to see if contains child pornography. There are a number of ways of doing that, the simplest involving automated tests to see if file descriptions contain keywords like 'child' or 'underage' or

something similar. In addition, our flagging system would have an option for child pornography under the explicit section, and videos flagged as such would be given the highest priority for review.

8.2 Economics

8.2.1 Project Costs

The only part of our project that cost us any money was the web hosting. However, we received two thousand dollars of credit from Google Cloud, which more than paid for the expenses we incurred.

8.2.2 Revenue Model

Divy has a B2C e-business model and is designed to as a transactional fee revenue model. It provides a digital environment where buyers and sellers can go for the transaction of goods. Divy's revenue comes from taking a small percentage of each purchase made on the site.

8.2.3 Future Revenue Sources

Though a basic transactional revenue model was initially desiged for Divy, there are a number of ways this can be expanded to allow for new revenue sources. The primary option would be to become a community provider which would allow the site to incorportate both advertising and a subscription revenue model. This can be done by allowing companies to purchase ad space on Divy's site. It also would allow for users to purchase a premium profile which would allow them to upload more content than a free acount. Users may also purchase sponsored posts which would ensure their work was showcased near the top of search results of specified keywords. Other ideas related to further developing Divy would require artists to sell a certain amount of work or hit a number of downloads to keep their account free, which would reduce the amount of clutter uploaded and ensure Divy was making money on each artist.

8.3 Safety

We need to do everything we can to ensure that our users are not exposed to any harmful software because of us. Thus, we have to check that users do not upload malicious files to our site. If we did absolutely no checking at all, it would be quite simple for a user to disguise a harmful program as something benign. Though such files might not hurt our own servers, they could wreak havoc upon a user's computer once downloaded. However, we can not use the same tactic we used to deal with copyright issues. It would not be okay for us to say that users themselves are responsible for making sure content they download is safe, and that it is their own responsibility to complain or take legal action against a user who uploaded malicious content. Instead, we need to make sure that each and every file uploaded to our servers could not harm once downloaded. We want our site to be safe, reliable, and easy to use, and it should be our responsibility to make sure that nobody's computer gets infected with malware because of us.

On the subject of security, we also need to make sure that the site itself is secure. We do want users to be able to charge and pay money for downloads, and we need to make sure that these transactions are secure. Otherwise, somebody's credit card information could be stolen, which would be quite awful. Thus, we have the responsibility to make sure this does not happen. In order to do this, we are planning on using PayPal, which has been in use for years and is widely trusted.

We will also do extensive vulnerability testing on the site, to make sure that nobody could break in and compromise our users' data.

Using Neo4j also provided us with hightened security as the graph database only has one known vulnerability and steps have been taken to rectify the possiblity of a security breach. This was yet another reason that this particular graph database was advantageous for our project.

8.4 Manufacturability

It was quite feasible to build our website. As was stated in previous sections of this document, each member of our team has extensive experience with web development and programming in general. In addition, there was nothing really new in our project - we were not doing anything that has not already been done many times with the exception of the recomendation system, which is commonly implemented on websites today but is not something any team members had previusly constructed.

8.5 Environmental Impact

Websites can consume a great deal of physical resources, and ours is no exception. We host our data on servers, all of which are located in a large data center somewhere. Data centers consume massive amounts of electricity, and as the content uploaded by users to our site grows, we'll be consuming more power. This is one of the reasons why we went with Google Cloud as our host. Google has invested a great deal of time and resources into research on sustainable energy, which shows that they care about environmental impact.

8.6 Usability

Usability is one of the most important aspects of a website, or, for that matter, any system. If users cannot figure out how to use our website, nobody will ever use it. Thus, we placed high priority on making our final product intuitive, straight-forward, and simple. We also went to great lengths to make our website conform to the government's standards for usability, and we considered the needs of disabled users, such as those who are color-blind, have poor eye sight, and so on, when designing our interfaces.

8.7 Lifelong Learning

Computing technologies evolve and are improved at lightning speed, and this is certainly true for the technologies used to build the web. We built our website using a few of the best technologies available to us today, but we realize that there will certainly be better tools in a few years' time. Thus, to keep our website from becoming outdated, we need to keep educating ourselves on what the latest, most innovative technologies are. If said technologies are a marked improvement over the languages and tools we used to build our website, then we will need to incorporate them, lest our work becomes obsolete.

9 User Agreement

When using the website, users will be subject to a terms of service that will cover the following legal issues. This is our complete User Agreement.

9.1 Service Agreement

A. By using or visiting the Divy website or any Divy products, software, data feeds, and services provided to you on, from, or through the Divy website, you signify your agreement to this User Agreement, and Divy's Community Guidelines. If you do not agree to any of these terms or the Community Guidelines, please do not use Divy.

B. These Terms of Service apply to all users of Divy.

C. You agree that Divy shall retain ten percent of all profits made through the website.

D. Pages on Divy may contain links to third party websites that are not owned or controlled by Divy. Divy has no control over, and assumes no responsibility for, the content, privacy policies, or practices of any third party websites. In addition, Divy will not and cannot censor or edit the content of any third-party site. By using this website, you expressly relieve Divy from any and all liability arising from your use of any third-party website.

9.2 User Content and Conduct

A. As the holder of an account on Divy, you may upload content, including videos, audio files, photographs, PDFs, user comments, and any other kind of digital file. You understand that Divy does not guarantee any confidentiality with respect to any Content you submit.

B. You shall be solely responsible for your own Content and the consequences of submitting and publishing your Content on Divy. You affirm, represent, and warrant that you own or have the necessary licenses, rights, consents, and permissions to publish Content you submit; and you license to Divy all patent, trademark, trade secret, copyright or other proprietary rights in and to such Content for publication on the Service pursuant to these Terms of Service.

C. You further agree that Content you submit will not contain third party copyrighted material, or material that is subject to other third party proprietary rights, unless you have permission from the rightful owner of the material or you are otherwise legally entitled to post the material and to grant Divy all of the license rights granted herein.

D. You further agree that you will not submit any Content or other material that is contrary to the Divy Community Guidelines, which may be updated from time to time, or contrary to applicable local, national, and international laws and regulations.

E. You further agree that your account on Divy may be suspended or banned if you are found in violation of Divy Community Guidelines, or of local, national, and international laws and regulations. Divy reserves the right to suspend and ban accounts without prior notice.

F. Divy does not endorse any Content submitted by any user or other licensor, or any opinion, recommendation, or advice expressed therein, and Divy expressly disclaims any and all liability

in connection with Content. Divy does not permit copyright infringing activities and infringement of intellectual property rights on the Service, and Divy will remove all Content if properly notified that such Content infringes on another's intellectual property rights. Divy reserves the right to remove Content without prior notice.

9.3 Use of Content

A. As the holder of an account on Divy, you may purchase and download any content on the site. Payments must be processed before content is downloaded.

B. You shall not copy, reproduce, distribute, transmit, broadcast, display, sell, license, or otherwise exploit any Content for any other purposes without the prior authorization of the respective licensors of the Content.

C. You agree not to access Content through any technology or means other than the ones provided by Divy.

D. You agree not to circumvent, disable or otherwise interfere with security-related features of the Service or features that prevent or restrict use or copying of any Content or enforce limitations on use of the Service or the Content therein.

E. You understand that when using Divy, you will be exposed to Content from a variety of sources, and that Divy is not responsible for the accuracy, usefulness, safety, or intellectual property rights of or relating to such Content. You further understand and acknowledge that you may be exposed to Content that is inaccurate, offensive, indecent, or objectionable, and you agree to waive, and hereby do waive, any legal or equitable rights or remedies you have or may have against Divy with respect thereto, and, to the extent permitted by applicable law, agree to indemnify and hold harmless Divy, its owners, operators, affiliates, licensors, and licensees to the fullest extent allowed by law regarding all matters related to your use of the website.

9.4 Digital Millennium Copyright Act

Any work found to be in violation of the Digital Millennium Copyright Act will be taken down. If the rightful oner wishes to pursue a leagal claim with the user who uploaded the stolen content is solely responsible and any legal action taken against them will not involve Divy in any way nor will Divy be expected to partake in the claim in any way.

10 Conclusion

10.1 Report Summary

In this report, we presented our design and first iteration for a website that allows users to upload any kind of digital content of their own creation, and specify the price for which it can be downloaded. The content exchange takes place within an online social community to help foster connections between artists and fans of their work. The decision to implement our system as a website, as opposed to a desktop application, came from finding many disadvantages with the latter. The construction of the website included the use of PHP, HTML5, CSS, JavaScript, CoffeeScript, and a graph database. In addition to these technologies, a clean modern and aesthetically pleasing design scheme was created for the site. Together, the design and technologies create an environment that allows users to easily navigate the site and complete actions such as uploading or downloading content, purchasing content, setting up an account and flagging content.

Beyond the design of the site, this report also focuses on the planning of how to complete the project and preventing any issues that may hinder the project from cropping up. Here a basic project timeline has been explained in addition to assigning roles to each team member. Mitigation strategies for preventing the team from failing to complete the project due to running out of time, inability to master new technologies, data loss or emergencies effecting team members have been thought of and implemented as well. These strategies and planning have been put into effect to allow the team to successfully meet all the functional and non-functional requirements that have been described for the site.

10.2 Lessons Learned

- Utalize Online Resources - Standard languages have a multitude of online resources, documentation, and forums online. We accepted a little to late that it is important to use these to speed one's learning and debugging process.
- Open Source frameworks - Using existing CSS frameworks, as opposed to attempting to build the design from the ground up, is key. We wanted complete control over the design but after months of work found that no matter what we did, our site seemed to lack the polish and details sites are expected to have in the current day and age. We leared that frameworks such as bootstrap allow for much quicker design overhauls and, becuase of the amount of time that has been dedicated to creating them, provide a much more complete look than what we could create with our resources.
- We spent much of the time early on creating different design options but not actually picking one to move forward with and to further develop. This slowed our design process, causing a delay in our progress on the backend functionality of the site. Learning poper time management is a huge take away from Divy.
- Standard languages - There are reasons these languages are standards, and we learned the hard way that we should trust the industry. We initially thought that a new language, called "Go" would be the most ideal for our project as it would provide a highly scalable framework that would maintain its speed as the site grew. However, we soon learned that it contains many obscurities and we would have to spend a lot longer learning it as compared to a more standardized lanuage.

- Use GitHub - this was a rather painful lesson that was learned in the early morning hours when an incorrect command or unfortunate accident caused the loss of hours of work. Backing up code and ensuring that there are multiple copies of work is the best way to prevent major issues from arising when new work is being done on the project.

10.3 Future Work

While Divy in its rudimentary form is complete there are countless ways for it to be further expanded especially if the project was to be turned into a company who wished the launch. There are a number of areas where work should be continued.

The first would be in expanding the social aspects of Divy. While it currently includes profiles and comments, private messaging, tagging, and writing on someone's profile should be built in if it is to be a viable social network. Logging in with existing soical networks would also increase the ease of registration for users. Allowing users to post directly to other social networks should also be built in to allow for Divy to become a part of the social networking landscape. Continuing with this would be to further build out how the sites revenue is collected and possibilities of reveue generation that were discussed in the future economics section of this report.

There is also work to be done on the recomendation system. A more comprehensive machine learning program should be written and tested as more users register and more content is uploaded. Further work could also be done to weigh ratings based on how users normally rate content, and more personalized recomendations can be made if more of the users' data is taken into account.

11 References

1. Bandcamp. (n.d.). Bandcamp. Retrieved October 10, 2013, from https://bandcamp.com/
2. Apple iTunes Everything you need to be entertained..Retrieved October 10, 2013, from http://www.apple.com/itunes/
3. Share Your Sounds. (n.d.). SoundCloud. Retrieved October 10, 2013, from http://soundcloud.com/
4. Spotify Music for Everyone. Spotify. Retrieved October 3, 2013, from http:spotify/com/us/
5. YouTube. (n.d.). YouTube. Retrieved October 10, 2013, from http://www.youtube.com/
6. PHP: Hypertext Preprocessor. (n.d.). PHP: Hypertext Preprocessor. Retrieved January 9, 2014, from http://www.php.net
7. HTML5. (n.d.). HTML5. Retrieved October 4, 2013, from http://www.w3.org/TR/html5/
8. Cascading Style Sheets home page. (n.d.). Cascading Style Sheets. Retrieved November 5, 2013, from http://www.w3.org/Style/CSS/Overview.en
9. JavaScript Tutorial. (n.d.). JavaScript Tutorial. Retrieved September 4, 2013, from http://www.w3schools.com/js/
10. JQuery. (n.d.). JQuery. Retrieved November 28, 2013, from http://jquery.com
11. Neo4j - The World's Leading Graph Database. (n.d.). Neo4j - The World's Leading Graph Database. Retrieved January 10, 2014, from http://www.neo4j.org
12. Classic API References. (n.d.). - PayPal Developer. Retrieved January 4, 2014, from https://developer.paypal.com/docs/classic/api/
13. Bootstrap (n.d.) Bootstrap. Retrieved Arpil 30, 2014, from http://getbootstrap.com
14. PHPUnit - Welcome to PHPUnit. (n.d.) PHPUnit. Retrieved April 14, 2014, from http://phpunit.de
15. Pivotal/jsunit.GitHub. Retrieved April 14,2014, from https://github.com/pivotal/jsunit
16. Selenium - Web Browser Automation (n.d.). Selenium. Retrieved April 14,2014, from http://docs.seleniumhq.org/
17. Google Cloud Platform - Cloud Computing and Cloud Hosting Services — Google Cloud Platform. (n.d.). Google Cloud Platform - Cloud Computing and Cloud Hosting Services — Google Cloud Platform. Retrieved February 4, 2014, from http://cloud.google.com/
18. Huberdeau Huberdeau L. 2014. Neo4j-PHP-OGM: Object Management Layer Built on Top of the neo4jphp Framework. Avaiable At: https://github.com/lphuberdeau/Neo4j-PHP-OGM [Accesed 24th May 2014]
19. Adell Adell J. 2014. neo4jphp: PHP Wrapper for the Neo4j graph database REST interface. Available At: https://github.com/jadell/neo4jphp [Accessed 21st May 2014]
20. Doctrine Doctrine2 Developers. 2014. Doctrine 2 ORM. Available At: https://github.com/doctrine/doctrine2 [Accessed 18th May 2014]. Doctrine2 Developers: https://github.com/orgs/doctrine/members

12 Appendix

12.1 Divy Neo4J-PHP-OGM Reference Manual

Divy Neo4JPHP-OGM Reference Manual

by

Aidan Crosbie, Lauren Falzarano, and Nicole Pal

Version 0.9.6
Last Edited: June 12, 2014

12.1.1 Introduction and Acknowledgements

Special thanks to Louis-Philippe Huberdeau[18] on GitHub for creating the Neo4J-PHP-OGM framework, to Josh Adell[19] on GitHub for creating the base Neo4JPHP framework, and the developers of Doctrine[20] for creating the Doctrine2 framework.

Neo4J-PHP-OGM extension allows for easy use to relate users, files, and comments. To keep the abundance of relationships in order, we've made this manual to explain the entities in the back-end, the cypher queries used to find these entities, and how to use the entities within the website itself. Actual usage of the entities and repositories can be seen directly in the source code. This is to give a comprehensive understanding of the Entities and Repositories of Divy.

12.1.2 Entities

Entites are essentially PHP Classes. We can use them to store relevant information in various nodes, whether it be a user email, a file type, or the number of likes a comment has. There are associated get, set, add, remove, and has functions. The following sections go over the different types of entities, the properties within them, and sample uses for interaction with that entities property.

12.1.2.1 User Entity

The User Entity Class has many properties in order to better recommend files and subscriptions to users. There are various property types that will be discussed in the following sections.

12.1.2.1.1 Indexes

User Indexes are used in searching and finding Users relatively quickly.

12.1.2.1.1.1 User Id

userId will store an internally created ID on node generation. By using OGM Auto, we can achieve this. userId has get and set functions.

Listing 1: User Id Index

```
/**********************************************************
* Variable: userId
**********************************************************
* Description: Stores the Users Id.
**********************************************************/
/**
* @OGM\Auto
*/
protected $userId;


/**********************************************************
* Function: getUserId()
**********************************************************
```

```
* Description: Returns the current users id.
*****************************************************/
function getUserId(){
        return $this->userId;
}


/*******************************************************
* Function: setUserId($userId)
*******************************************************
* Description: Sets the current users id.
*******************************************************/
function setUserId($userId)
        $this->userId=$userId;
}


/*******************************************************
* Example Description: Retrieve the Users Id, then set the
* Users Id with the retrieved value.
*******************************************************/
$userId = $thisUser->getUserId();
$thisUser->setUserId($setUserId);
```

12.1.2.1.1.2 User Unique Id

userUniqueId is automatically generated in the construct function. userUniqueId only has a get function.

Listing 2: User Unique Id Index

```
/*******************************************************
* Variable: userUnqiueId
*******************************************************
* Description: Stores the users unique id.
*******************************************************/
/**
* @ODM\Property
* @ODM\Index
*/
protected $userUniqueId;


/*******************************************************
* Function: getUserUniqueId()
*******************************************************
* Description: getUserUniqueId() will return the users
```

```
* unique id.
*********************************************************/
fucntion getUserUniqueId(){
        return $this->userUniqueId;
}


/*********************************************************
* Example Description: Retrieve the users unique id.
*********************************************************/
$thisUserUniqueId = $thisUser->getUserUniqueId();
```

12.1.2.1.1.3 User Name

userName will be indexed for faster searching, but will simply have the name of the user. userName has get and set functions.

Listing 3: User Name Index

```
/*********************************************************
* Variable: userName
*********************************************************
* Description: Stores the users name
*********************************************************/
/**
* @OGM\Property
* @OGM\Index
*/
protected $userName;


/*********************************************************
* Function: getUserName()
*********************************************************
* Description: getUserName will return the users name
*********************************************************/
function getUserName(){
        return $this->userName;
}


/*********************************************************
* Function: setUserName($userName)
*********************************************************
* Description: Sets the current user name
*********************************************************/
function setUserName($userName){
```

```
        $this->userName = $userName;
}


/*******************************************************
* Example Description:Retrieve the old users name and then
* update the users name.
*******************************************************/
$oldUserName = $thisUser->getUserName();
$newUserName = 'New Name';
$thisUser->setUserName($newUserName);
```

12.1.2.1.2 Properties

User Properties hold a variety of user information, ranging from the users password, to a file they wish to feature on their page.

12.1.2.1.2.1 User First Name

userFName will simply have the users first name. userFname has get and set functions.

Listing 4: User First Name Property

```
/*******************************************************
* Variable: userFName
*******************************************************
* Description: Stores the users first name
*******************************************************/
/**
* @ODM\Property
*/
protected $userFName;


/*******************************************************
* Function: getUserFName()
*******************************************************
* Description: Returns the current users first name
*******************************************************/
function getUserFName(){
        return $this->userFName;
}


/*******************************************************
* Function: setUserFName($userFName)
*******************************************************
* Description: Sets the current users first name
```

```
*******************************************************/
function setUserFName($userFName){
        $this->userFName = $userFName;
}


/*********************************************************
* Example Description: Returns the users old first name,
* then sets the users new first name
*********************************************************/
$oldFirstName = $thisUser->getUserFName();
$newFirstName = 'Steve';
$thisUser->setUserFName($newFirstName);
```

12.1.2.1.2.2 User Last Name

userLName will simply have the users last name. userLName has get and set functions.

Listing 5: User Last Name Property

```
/*********************************************************
* Variable: userLName
*********************************************************
* Description: Stores the users last name
*********************************************************/
/**
* @OGM\Property
*/
protected $userLName;


/*********************************************************
* Function: getUserLName()
*********************************************************
* Description: Returns the current users last name
*********************************************************/
function getUserLName(){
        return $this->userLName;
}


/*********************************************************
* Function: setUserLName($userLName)
*********************************************************
* Description: Sets the current users last name
*********************************************************/
function setUserLName($userLName){
```

```
        $this->userLName = $userLName;
}


/*************************************************************
* Example Description: Retrieves the users old last name,
* then sets the users new last name
*************************************************************/
$oldLastName = $thisUser->getUserLName();
$newLastName = 'Janowski';
$thisUser->setUserLName($newLastName);
```

12.1.2.1.2.3 User Email

userEmail will simply store the users email. userEmail has get and set functions.

Listing 6: User Email

```
/*************************************************************
* Variable: userEmail
*************************************************************
* Description: Stores the users email
*************************************************************/
/**
* @OGM\Property
*/
protected $userEmail;


/*************************************************************
* Function: getUserEmail()
*************************************************************
* Description: Returns the current users email
*************************************************************/
function getUserEmail(){
        return $this->userEmail;
}


/*************************************************************
* Function: setUserEmail($userEmail)
*************************************************************
* Description: Sets the current users email
*************************************************************/
function setUserEmail($userEmail){
        $this->userEmail = $userEmail;
}
```

```
/*********************************************************
* Example Description: Retrieve the old users email,
* and sets the users new email
*********************************************************/
$userOldEmail = $thisUser->getUserEmail();
$newUserEmail = 'StevieJanowski@MertleBeachMermen.com';
$thisUser->setUserEmail($newUserEmail);
```

12.1.2.1.2.4 User Password

userPassword will store an md5 encrypted password entered by the user. userPassword has get and set functions.

Listing 7: User Password Property

```
/*********************************************************
* Variable: userPassword
*********************************************************
* Description: The users md5 encrypted password
*********************************************************/
/**
* @OGM\Property
*/
protected $userPassword;


/*********************************************************
* Function: getUserPassword()
*********************************************************
* Description: Returns the current users md5 password
*********************************************************/
function getUserPassword(){
        return $this->userPassword;
}


/*********************************************************
* Function: setUserPassword($userPassword)
*********************************************************
* Description: Sets the current users md5 password
*********************************************************/
function setUserPassword($userPassword){
        $this->userPassword = md5($userPassword);
}
```

```
/*********************************************************
* Example Description: Get the old user password and set the
* new user password
*********************************************************/
$oldMd5Password = getUserPassword();
$newPassword = 'KennyPowersRules';
$thisUser->setUserPassword($newPassword);
```

12.1.2.1.2.5 User Autenticated

userAuth will store a boolean value to whether the user has confirmed there account by our email confirmation yet. userAuthenticated has get and set functions.

Listing 8: User Authenticated Property

```
/*********************************************************
* Variable: userAuth
*********************************************************
* Description: Stores if the user is authenticated to
* the site
*********************************************************/
/**
* @OGM\Property
*/
protected $userAuth;


/*********************************************************
* Function: getUserAuth()
*********************************************************
* Description: Returns if the user is authenticated
*********************************************************/
function getUserAuth(){
        return $this->userAuth;
}


/*********************************************************
* Function: setUserAuth($userAuth)
*********************************************************
* Description: Sets the new boolean value to if the user
* is authenticated yet
*********************************************************/
function setUserAuth($userAuth){
        $this->userAuth = $userAuth;
}
```

```
/*********************************************************
* Example Description: Checks if the user is already
* authenticated, if not we make the user authenticated.
*********************************************************/
if($thisUser->getUserAuth() == false){
        $thisUser->setUserAuth(true);
}
```

12.1.2.1.2.6 User Authentication Code

userAuthCode is a randomly generated string that will be used to verify users by email confirmation. userAuthCode has get and set functions.

Listing 9: User Authentication Code Property

```
/*********************************************************
* Variable: userAuthCode
*********************************************************
* Description: Stores the authentication code for user
* authentication.
*********************************************************/
/**
* @OGM\Property
*/
protected $userAuthCode;


/*********************************************************
* Function: getUserAuthCode()
*********************************************************
* Description: Returns the users authentication code.
*********************************************************/
function getUserAuthCode(){
        return $this->userAuthCode;
}


/*********************************************************
* Function: setUserAuthCode()
*********************************************************
* Description: Sets the users authentication code.
*********************************************************/
function setUserAuthCode($userAuthCode){
        $this->userAuthCode = $userAuthCode;
}
```

```
/*********************************************************
* Example Description: Sets the users authentication code,
* then will get the authentication code.
*********************************************************/
$thisUser->setUserAuthCode('ABC123');
$thisUserAuthCode = $thisUser->getUserAuthCode();
```

12.1.2.1.2.7 User Admin Level

userAdminLevel stores if the current user has administrative properties. This can be to editing files, deleting files, and deleting comments. userAdminLevel has get and set functions.

Listing 10: User Admin Level Property

```
/*********************************************************
* Variable: userAdminLevel
*********************************************************
* Description: Stores if the user has administrative
* properties
*********************************************************/
/**
* @OGM\Property
*/
protected userAdminLevel;


/*********************************************************
* Function: getUserAdminLevel()
*********************************************************
* Description: Returns the users admin level
*********************************************************/
function getUserAdminLevel(){
        return $this->userAdminLevel;
}


/*********************************************************
* Function: setUserAdminLevel($userAdminLevel)
*********************************************************
* Description: Sets the users admin level
*********************************************************/
function setUserAdminLevel($userAdminLevel){
        $this->userAdminLevel = $userAdminLevel;
}
```

```
/*********************************************************
* Example Description: Checks to see if the user has any
* adminstrative properties, then gives them all
* administrative properties.
*********************************************************/
$oldUserAdminLevel = $thisUser->getUserAdminLevel();
if($oldUserAdminLevel == 'None'){
        $thisUser->setUserAdminLevel('DivyDev');
}
```

12.1.2.1.2.8 User Age

userAge is stored as a time stamp and will be used to verify age quickly for explicit files. Further age authentication will be implemented when an automated credit card process is added to the site. userAge will have get and set functions.

Listing 11: User Age Property

```
/*********************************************************
* Variable: userAge
*********************************************************
* Description: Stores the users age.
*********************************************************/
/**
* @OGM\Property(format="date")
*/
protected userAge;


/*********************************************************
* Function: getUserAge()
*********************************************************
* Description: Returns the users age
*********************************************************/
function getUserAge(){
        return $this->userAge;
}


/*********************************************************
* Function: setUserAge($userAge)
*********************************************************
* Description: Sets the users age
*********************************************************/
function setUserAge($userAge){
        $this->userAge = $userAge;
```

```
}


/*********************************************************
* Example Description: Get the user age, and set their new
* age to today.
*********************************************************/
$oldUserAge = $thisUser->getUserAge();
$newUserAge = date("Y-m-d H:i:s");
$thisUser->setUserAge($newUserAge);
```

12.1.2.1.2.9 User Register Date

userRegisterDate is stored as a time stamp and will be used to distinguish between newer and older users. userRegisterDate has get and set functions.

Listing 12: User Register Date Property

```
/*********************************************************
* Variable: userRegisterDate
*********************************************************
* Description: Stores the date the user registered
*********************************************************/
/**
* @OGM\Property(format="date")
*/
protected userRegisterDate;


/*********************************************************
* Function: getUserRegisterDate()
*********************************************************
* Description: Returns the date the user registered
*********************************************************/
function getUserRegisterDate(){
        return $this->userRegisterDate;
}


/*********************************************************
* Function: setUserRegisterDate($userRegisterDate)
*********************************************************
* Description: Sets the date the user registered
*********************************************************/
function setUserRegisterDate($userRegisterDate){
        $this->userRegisterDate = $userRegisterDate;
}
```

```
/*********************************************************
* Example Description: Set the users registered date to
* today, and then get that date.
*********************************************************/
$newRegisterDate = date("Y-m-d H:i:s");
$thisUser->setUserRegisterDate($newRegisterDate);
```

12.1.2.1.2.10 User Profile Photo

userProfilePhoto stores the name of the profile picture used by the user. We could potentially store this as a blob in the node, but this is known to cause serious issues in node retrieval. userProfilePhoto has get and set functions

Listing 13: User Profile Photo Property

```
/*********************************************************
* Variable: userProfilePhoto
*********************************************************
* Description: Stores the name of the user profile photo
*********************************************************/
/**
* @OGM\Property
*/
protected userProfilePhoto;


/*********************************************************
* Function: getUserProfilePhoto()
*********************************************************
* Description: Returns the name of the users profile photo
*********************************************************/
function getUserProfilePhoto(){
        return $this->userProfilePhoto;
}


/*********************************************************
* Function: setUserProfilePhoto($userProfilePhoto)
*********************************************************
* Description: Sets the name of the user profile photo
*********************************************************/
function setUserProfilePhoto($userProfilePhoto){
        $this->userProfilePhoto = $userProfilePhoto;
}
```

```
/*********************************************************
* Example Description: Set the new user photo name, and then
* get the location of this photo in our directories.
*********************************************************/
$newUserPhoto = 'MyGoodTimes.jpg';
$thisUser->setUserProfilePhoto($newUserPhoto);
$userName = $thisUser->getUserName();
$newUserPhotoDir = '/images/'.$userName.'/'.$newUserPhoto.'';
```

12.1.2.1.2.11 User Css

userCss will store the custom css file name the user has selected. userCss hass get and set functions

Listing 14: User Css Property

```
/*********************************************************
* Variable: userCss
*********************************************************
* Description: Stores the users selected css
*********************************************************/
/**
* @ODM\Property
*/
protected userCss;


/*********************************************************
* Function: getUserCss()
*********************************************************
* Description: Returns the users selected Css
*********************************************************/
public function getUserCss(){
        return $this->userFeaturedCss;
}


/*********************************************************
* Function: setUserCss($userCss)
*********************************************************
* Description: Sets the users selected css
*********************************************************/
public function setUserCss(File $userCss){
        $this->userCss = $userCss;
}
```

```
/*********************************************************
* Example Description: Get the users css, and if its set
* to standard, select custom css.
*********************************************************/
$oldCss = $thisUser->getUserCss();
if($oldCss == 'Standard'){
        $newCss = 'Custom.css';
        $thisUser->setUserCss($newCss);
}
```

12.1.2.1.2.12 User Featured File

userFeatured file will actually store a file node. This is so users are able to promote files they wish to. We see on occassion that an artist has a work their proud of, but there other works receive more attention. userFeaturedFile has get and set functions.

Listing 15: User Featured File Property

```
/*********************************************************
* Variable: userFeaturedFile
*********************************************************
* Description: Stores the users featured file.
*********************************************************/
/**
* @ODM\ManyToOne(relation="featuredUserFile")
*/
protected userFeaturedFile;


/*********************************************************
* Function: getUserFeaturedFile()
*********************************************************
* Description: Returns the users featured file.
*********************************************************/
public function getUserFeaturedFile(){
        return $this->userFeaturedFile;
}


/*********************************************************
* Function: setUserFeaturedFile(File $userFeaturedFile)
*********************************************************
* Description: Sets the users featured file
*********************************************************/
public function setUserFeaturedFile(File $userFeaturedFile){
        $this->userFeaturedFile = $userFeaturedFile;
}
```

```
/*********************************************************
* Example Description: Get the old featured file. If this
* file name does not match the id given, set the new
* featured file
*********************************************************/
$BobsAwesomeNewFile = new File;
$oldFeatured = $thisUser->getUserFeaturedFile();
if($oldFeatured->getFileName() == 'Bobs Old Overrated File'){
        $thisUser->setFeaturedFile($BobsAwesomeNewFile);
}
```

12.1.2.1.2.13 User Total Owned File Likes

userTotalOwnedFileLikes will store the total number of file likes a user has. userTotalOwnedFileLikes has get and set functions.

Listing 16: User Total Owned File Likes Property

```
/*********************************************************
* Variable: userTotalOwnedFileLikes
*********************************************************
* Description: Stores the number of file likes a user has
*********************************************************/
/**
* @ODM\Property
*/
protected userTotalOwnedFileLikes;


/*********************************************************
* Function: getUserTotalOwnedFileLikes()
*********************************************************
* Description: Returns the number of file likes a user has
*********************************************************/
function getUserTotalOwnedFileLikes(){
        return $this->userTotalOwnedFileLikes;
}


/*********************************************************
* Function:
* setUserTotalOwnedFileLikes($userTotalOwnedFileLikes)
*********************************************************
* Description: Sets the number of file likes a user has
*********************************************************/
```

```
function setUserTotalOwnedFileLikes($userTotalOwnedFileLikes){
        $this->userTotalOwnedFileLikes = $userTotalOwnedFileLikes;
}


/*********************************************************
* Example Description: Get the old file likes, and if the
* user has none give them 100 for no apparent reason.
*********************************************************/
$oldFileLikes = $thisUser->getUserTotalOwnedFileLikes();
if($oldFileLikes == 0){
        $newFileLikes = 100;
        $thisUser->setUserTotalOwnedFileLikes($newFileLikes);
}
```

12.1.2.1.2.14 User Total Owned File Dislikes

userTotalOwnedFileDislikes will store the number of file dislikes a user has. userTotalOwnedFileDislikes has get and set functions

Listing 17: User

```
/*********************************************************
* Variable: userTotalOwnedFileDislikes
*********************************************************
* Description: Stores the number of file dislikes a user has
*********************************************************/
/**
* @OGM\Property
*/
protected userTotalOwnedFileDislikes;


/*********************************************************
* Function: getUserTotalOwnedFileDislikes()
*********************************************************
* Description: Returns the number of file dislikes a user
* has
*********************************************************/
function getUserTotalOwnedFileDislikes(){
        return $this->userTotalOwnedFileDislikes;
}


/*********************************************************
* Function:
* setUserTotalOwnedFileDislikes($userTotalOwnedFileDislikes)
```

```
*********************************************************
* Description: Sets the number of file dislikes a user has
*********************************************************/
function setUserTotalOwnedFileDislikes($userTotalOwnedFileDislikes){
        $this->userTotalOwnedFileDislikes = $userTotalOwnedFileDislikes
            ;
}


/*********************************************************
* Example Description: Get the users file dislikes, and there
* are any make set the user file dislikes to 0 for no
* apparent reason.
*********************************************************/
$fileDislikes = $thisUser->getUserTotalOwnedFileDislikes();
if($fileDislikes > 0){
        $newFileDislikes = 0;
    $thisUser->setUserTotalOwnedFileDislikes($newFileDislikes);
}
```

12.1.2.1.2.15 User Total Owned File Views

userTotalOwnedFileViews will store the number of file views a user has. userTotalOwnedFileViews has get and set functions.

Listing 18: User Total Owned File Views Property

```
/*********************************************************
* Variable: userTotalOwnedFileViews
*********************************************************
* Description: Stores the number of file views a user has
*********************************************************/
/**
* @ODM\Property
*/
protected userTotalOwnedFileViews;


/*********************************************************
* Function: getUserTotalOwnedFileViews()
*********************************************************
* Description: Returns the number of file views a user has
*********************************************************/
function getUserTotalOwnedFileViews(){
        return $this->userTotalOwnedFileViews;
}
```

```
/*********************************************************
* Function:
* setUserTotalOwnedFileViews($userTotalOwnedFileViews)
*********************************************************
* Description: Sets the number of file views a user has.
*********************************************************/
function setUserTotalOwnedFileViews($userTotalOwnedFileViews){
        $this->userTotalOwnedFileViews = $userTotalOwnedFileViews;
}


/*********************************************************
* Example Description: Gets the old user file views.
* Will reset the viewes if they are over one hundred million
* for no apparent reason.
*********************************************************/
$fileViews = $thisUser->getUserTotalOwnedFileViews();
if($fileViews >= $oneHundredMillion){
        $newFileViews = 0;
        $thisUser->setUserTotalOwnedFileViews($newFileViews);
}
```

12.1.2.1.2.16 User Total Owned File Purchases

userTotalOwnedFilePurchases will store how many files a user has sold. userTotalOwnedFilePurchases has get and set functions.

Listing 19: User Owned File Purchases

```
/*********************************************************
* Variable: userTotalOwnedFilePurchases
*********************************************************
* Description: Stores the number of files a user has sold
*********************************************************/
/**
* @OGM\Property
*/
protected $userTotalOwnedFilePurchases;


/*********************************************************
* Function: getUserTotalOwnedFilePurchases()
*********************************************************
* Description: Returns the number of files a user has sold
*********************************************************/
function getUserTotalOwnedFilePurchases(){
```

```
        return $this->userTotalOwnedFilePurchases();
}


/*********************************************************
* Function:
*setUserTotalOwnedFilePurchases($userTotalOwnedFilePurchases)
*********************************************************
* Description: Sets the number of files a user has sold
*********************************************************/
function setUserTotalOwnedFilePurchases($userTotalOwnedFilePurchases){
        $this->userTotalOwnedFilePurchases =
            $userTotalOwnedFilePurchases;
}


/*********************************************************
* Example Description: Gets the number of files a user has
* sold, if they have sold over one hundred thousand, remove
* a single file purchase for no apparent reason.
*********************************************************/
$filePurchases = $thisUser->getUserTotalOwnedFilePurchases();
$oneHundredThousand = 100000;
if($filePurchases > $oneHundredThousand){
        $subtractOne = $filePurchases -1;
        $thisUser->setUserTotalOwnedFilePurchases($substractOne);
}
```

12.1.2.1.2.17 User Total Subscriptions Made

userTotalSubscribedTo will store how many user subscriptions have been made. userTotalSubscribedTo has get and set functions.

Listing 20: User Total Subscribed To Property

```
/*********************************************************
* Variable: userTotalSubscribedTo
*********************************************************
* Description: Stores the number of user subscriptions
*********************************************************/
/**
* @OGM\Property
*/
protected userTotalSubscribedTo;


/*********************************************************
```

```
* Function: getUserTotalSubscribedTo()
*********************************************************
* Description: Returns the number of user subscriptions
*********************************************************/
function getUserTotalSubscribedTo(){
        return $this->userTotalSubscribedTo;
}


/*********************************************************
* Function: setUserTotalSubscribedTo($userTotalSubscribedTo)
*********************************************************
* Description: Sets the number of user subscriptions.
*********************************************************/
function setUserTotalSubscribedTo($userTotalSubscribedTo){
        $this->userTotalSubscribedTo = $userTotalSubscribedTo;
}


/*********************************************************
* Example Description: Gets the users subscription count.
* If the count is none, add one subsciption for no apparent
* reason.
*********************************************************/
$subscribedToCount = $thisUser->getUserTotalSubscribedTo();
if($subscribedToCount == 0){
        $newSubscribedTo = $subscribedToCount+1;
        $thisUser->setUserTotalSubscribedTo($newSubscribedTo);
}
```

12.1.2.1.2.18 User Total Subscribers

userTotalSubscribers will store how many users have made a subscription to the current user. userTotalSubscribes has get and set functions.

Listing 21: User Total Subscribers Property

```
/*********************************************************
* Variable: userTotalSubscribers
*********************************************************
* Description: Stores how many users have subscribed to
* this user.
*********************************************************/
/**
* @OGM\Property
*/
protected $userTotalSubscribers;
```

```
/*********************************************************
* Function: getUserTotalSubscribers()
*********************************************************
* Description: Returns the number of subscribers this
* user has
*********************************************************/
function getUserTotalSubscribers(){
        return $this->userTotalSubscribers;
}


/*********************************************************
* Function: setUserTotalSubscribers($userTotalSubscribers)
*********************************************************
* Description: Sets the total number of user subscribers
*********************************************************/
function setUserTotalSubscribers($userTotalSubscribers){
        $this->userTotalSubscribers = $userTotalSubscribers;
}


/*********************************************************
* Example Description: Get the number of subscribers a user
* has. If they have none, add one for no apparent reason.
*********************************************************/
$subscriberCount = $thisUser->getUserTotalSubscribers();
if($subscriberCount == 0){
        $newSubscribers = $subscriberCount+1;
    $thisUser->setUserTotalSubscribers($newSubscribers);
}
```

12.1.2.1.2.19 User Total Profile Comments

userTotalProfileComments will store the number of comments on this users profile page. userTotalProfileComments has get and set functions.

Listing 22: User Total Profile Comments Property

```
/*********************************************************
* Variable: userTotalProfileComments
*********************************************************
* Description: Stores the number of comments on this users
* profile page
*********************************************************/
/**
```

```
* @ODM\Property
*/
protected userTotalProfileComments;


/**********************************************************
* Function: getUserTotalProfileComments()
**********************************************************
* Description: Returns the number of profile comments on
* a users profile page.
**********************************************************/
function getUserTotalProfileComments(){
        return $this->userTotalProfileComments;
}


/**********************************************************
* Function:
* setUserTotalProfileComments($userTotalProfileComments)
**********************************************************
* Description: Sets the number of comments on a users
* profile page.
**********************************************************/
function setUserTotalProfileComments($userTotalProfileComments){
        $this->userTotalProfileComments = $userTotalProfileComments;
}


/**********************************************************
* Example Description: Get the users profile comment count.
* If there are none, add one hundred to cheer them up for
* no apparent reason.
**********************************************************/
$userCommentCount = $thisUser->getUserTotalProfileComments();
if($userCommentCount == 0){
        $newComments = $userCommentCount+100;
        $thisUser->setUserTotalProfileComments($newComments);
}
```

12.1.2.1.2.20 User Total Comments

userTotalComments will store the number of comments a user has made. userTotalComments has get and set functions.

Listing 23: User Total Comments Property

```
/**********************************************************
```

```
* Variable: userTotalComments
*****************************************************
* Description: Stores the number of comments a user has made
*****************************************************/
/**
* @ODM\Property
*/
protected userTotalComments;


/*****************************************************
* Function: getUserTotalComments()
*****************************************************
* Description: Returns the number of comments a user has made
*****************************************************/
function getUserTotalComments(){
        return $this->userTotalComments
}


/*****************************************************
* Function: setUserTotalComments($userTotalComments)
*****************************************************
* Description: Sets the number of comments a user has made
*****************************************************/
function setUserTotalComments($userTotalComments){
        $this->userTotalComments= $userTotalComments;
}


/*****************************************************
* Example Description: Get the old comment number, and then
* increment it by 5 for no apparent reason.
*****************************************************/
$countUserComments = $thisUser->getUserTotalComments();
$newCount = $countUserComments+5;
$thisUser->setUserTotalComments($newCount);
```

12.1.2.1.2.21 User Total Comment Likes

userTotalComment likes will store the number of likes a users comments have. userTotalCommentLikes has get and set functions.

Listing 24: User Total Comment Likes Property

```
/*****************************************************
* Variable: userTotalCommentLikes
```

```
/**********************************************************
* Description: Stores the number of likes this users
* comments have.
**********************************************************/
/**
* @OGM\Property
*/
protected $userTotalCommentLikes;


/**********************************************************
* Function: getUserTotalCommentLikes()
**********************************************************
* Description: Returns the total number of comment likes
* this user has.
**********************************************************/
function getUserTotalCommentLikes(){
        return $this->userTotalCommentLikes;
}


/**********************************************************
* Function: setUserTotalCommentLikes($userTotalCommentLikes)
**********************************************************
* Description: Sets the total number of comment likes this
* user has.
**********************************************************/
function setUserTotalCommentLikes($userTotalCommentLikes){
        $this->userTotalCommentLikes = $userTotalCommentLikes;
}


/**********************************************************
* Example Description: Get the old number of likes a user has
* and then increment it by 1.
**********************************************************/
$userCommentLikes = $thisUser->getUserTotalCommentLikes();
$newLikes = $userCommentLikes+1;
$thisUser->setUserTotalCommentLikes($newLikes);
```

12.1.2.1.2.22 User Total Comment Dislikes

userTotalCommentDislikes will store the total number of dislikes a users comments have. userTotalCommentDislikes has get and set functions.

Listing 25: User Total Comment Dislikes Property

```
/*********************************************************
* Variable: userTotalCommentDislikes
*********************************************************
* Description: Stores the total number of comment dislikes
* this user has.
*********************************************************/
/**
* @ODM\Property
*/
protected $userTotalCommentDislikes;


/*********************************************************
* Function: getUserTotalCommentDislikes()
*********************************************************
* Description: Returns the number of comment dislikes a
* user has.
*********************************************************/
function getUserTotalCommentDislikes(){
        return $this->userTotalCommentDislikes;
}


/*********************************************************
* Function: setUserTotalCommentDislikes($userTotalCommentDislikes)
*********************************************************
* Description: Sets the number of comment dislikes a user
* has.
*********************************************************/
function setUserTotalProfileComments($userTotalProfileComments){
        $this->userTotalProfileComments = $userTotalProfileComments;
}


/*********************************************************
* Example Description: Get the users old comment dislikes.
* If the user have more than 5 dislikes, we will remove
* one dislike.
*********************************************************/
$countCommentDislikes= $thisUser->getUserTotalCommentDislikes();
if($countCommentDislikes > 5){
        $newCount = $countCommentDislikes-1;
    $thisUser->setUserTotalCommentDislikes($newCount);
}
```

12.1.2.1.3 Arrays

There are no array properties in the User Class. They exist only in the file class.

12.1.2.1.4 ArrayCollections

User ArrayCollections hold an array of nodes relevant to the user. This could be the files that they liked, or the comments they have made.

12.1.2.1.4.1 User File Likes

userFileLikes will store an Array Collection of all the files a user liked. userFileLikes has get, set, add, remove, and has functions.

Listing 26: User File Likes Array Collection

```
/**********************************************************
* Variable: userFileLikes
**********************************************************
* Description: An ArrayCollection of the files the user
* liked.
**********************************************************/
/**
* @ODM\ManyToMany(relation="likedFile")
*/
protected $userFileLikes;


/**********************************************************
* Function: getUserFileLikes()
**********************************************************
* Description: Returns an array collection of the files the
* user liked.
**********************************************************/
function getUserFileLikes(){
        return $this->userFileLikes;
}


/**********************************************************
* Function: setUserFileLikes(ArrayCollection $userFileLikes)
**********************************************************
* Description: Sets the ArrayCollection of files the user
* liked.
**********************************************************/
function setUserFileLikes(ArrayCollection $userFileLikes){
        $this->userFileLikes = $userFileLikes;
}


/**********************************************************
```

```
* Function: addUserFileLikes(File $file)
*********************************************************
* Description: Adds a new file that a user liked to the
* ArrayCollection
*********************************************************/
function addUserFileLikes(File $file){
        $this->userFileLikes->add($file);
}


/*********************************************************
* Function: removeUserFileLikes(File $file)
*********************************************************
* Description: Removes a file that a user once liked from
* the ArrayCollection
*********************************************************/
function removeUserFileLikes(File $file){
        $this->userFileLikes->removeElement($file);
}


/*********************************************************
* Function: hasUserFileLikes(File $file)
*********************************************************
* Description: Checks to see if the file passed was liked
* by the user
*********************************************************/
function hasUserFileLikes(File $file){
        return $this->userFileLikes->contains($user);
}


/*********************************************************
* Example Description: Add a file like, remove that file like,
* set the new files liked, get those file liked, and check
* if the user liked the file.
*********************************************************/
$file = new File;

$hesitantHarry = $thisUser;

$hesistantHarry->addUserFileLikes($file);
$hesistantHarry->removeUserFileLikes($file);

$userFileLikes = $hesistantHarry->getUserFileLikes();
$hesitentHarry->setUserFileLikes($userFileLikes);
```

```
$didHarryLike = $hesitantHarry->hasUserFileLikes($file);
```

12.1.2.1.4.2 User File Views

userFileViews stores an ArrayCollection of files that the user has viewed. userFileViews has get, set, add, remove, and has functions.

Listing 27: User File Views Array Collection

```
/**************************************************************
* Variable: userFileViews
**************************************************************
* Description: Stores the ArrayCollection of files the user
* has viewed
**************************************************************/
/**
* @ODM\ManyToMany(relation="viewedFile")
*/
protected $userFileViews;


/**************************************************************
* Function: getUserFileViews()
**************************************************************
* Description: Returns the ArrayCollection of files the user
* has viewed
**************************************************************/
function getUserFileViews(){
        return $this->userFileViews;
}


/**************************************************************
* Function: setUserFileViews(ArrayCollection $userFileViews)
**************************************************************
* Description: Sets the ArrayCollection of files the user
* has viewed
**************************************************************/
function setUserFileViews(ArrayCollection $userFileViews){
        $this->userFileViews = $userFileViews;
}


/**************************************************************
* Function: addUserFileViews(File $file)
**************************************************************
```

```
* Description: Adds a file a user has viewed
*******************************************************/
function addUserFileViews(File $file){
        $this->userFileViews->add($file);
}


/*******************************************************
* Function: removeUserFileViews(File $file)
*******************************************************
* Description: Remove a file that a user viewed
*******************************************************/
function removerUserFileViews(File $file){
        $this->userFileViews->add($file)
}


/*******************************************************
* Function: hasUserFileViews(File $file)
*******************************************************
* Description: Check to see if the user has viewed the file
*******************************************************/
function hasUserFileViews(File $file){
        return $this->userFileViews->contains($file);
}


/*******************************************************
* Example Description: Add a user file view, remove a user
* file view, add a user file view, get the user file views,
* set the user file views, and then check if the user viewed
* the file.
*******************************************************/
$indecisiveIan = $thisUser;
$file = new File;

$indecisiveIan->addUserFileViews($file);
$indecisiveIan->removeUserFileViews($file);
$indicisiveIan->addUserFileViews($file);

$iansViews = $indecisiveIan->getUserFileViews();
$indecisiveIan->setUserFileViews($iansViews);

$didIanView = $indecisiveIan->hasUserFileViews($file);
```

12.1.2.1.4.3 User File Dislikes

userFileDislikes will store an ArrayCollection of the files a user disliked.

Listing 28: User File Dislikes Array Collection

```
/**********************************************************
* Variable: userFileDislikes
**********************************************************
* Description: Stores the ArrayCollection of files the
* user disliked
**********************************************************/
/**
* @OGM\ManyToMany(relation="dislikedFile")
*/
protected $userFileDislikes;


/**********************************************************
* Function: getUserFileDislikes()
**********************************************************
* Description: Returns the files the user disliked
**********************************************************/
function getUserFileDislikes(){
        return $this->userFileDislikes;
}


/**********************************************************
* Function: setUserFileDislikes(ArrayCollection $userFileDislikes)
**********************************************************
* Description: Sets the files the user disliked
**********************************************************/
function setUserFileDislikes(ArrayCollection $userFileDislikes){
        $this->userFileDislikes = $userFileDislikes;
}


/**********************************************************
* Function: addUserFileDislikes(File $file)
**********************************************************
* Description: Adds a file that the user disliked
**********************************************************/
function addUserFileDislikes(File $file){
        $this->userFileDislikes->add($file);
}
```

```
/*******************************************************
* Function: removeUserFileDislikes(File $file)
*******************************************************
* Description: Removes a file that a user disliked
*******************************************************/
function removeUserFileDislikes(File $file){
        $this->userFileDislikes->removeElement($file);
}


/*******************************************************
* Function: hasUserFileDislikes(File $file)
*******************************************************
* Description: Checks if the user has viewed the file
*******************************************************/
function hasUserFileDislikes(File $file){
        return $this->userFileDislikes->contains($file);
}


/*******************************************************
* Example Description: Remove a user file dislike, add a user
* file dislike, get the files the user disliked, set the
* files the user disliked, and then check if the user
* disliked the file.
*******************************************************/
$displeasedDave = $thisUser;
$file = new File;
$displeasedDave->removeUserFileDislikes($file);
$displeasedDave->addUserFileDislikes($file);

$filesDislikedByDave = $displeasedDave->getUserFileDislikes();
$displeasedDave->setUserFileDislikes($fileDislikedByDave);

$didDaveDislike = $displeasedDave->hasUserFileDislikes($file);
```

12.1.2.1.4.4 User File Purchases

userFilePurchases stores an ArrayCollection of the files that a user has purchased. userFilePurchases has get, set, add, remove, and has functions.

Listing 29: User File Purchases Array Collection

```
/*******************************************************
* Variable: userFilePurchases
*******************************************************
* Description: Stores the files that a user has purchased
```

```
*********************************************************/
/**
* @OGM\ManyToMany(relation="purchasedFile")
*/
protected $userFilePurchases


/*********************************************************
* Function: getUserFilePurchases()
*********************************************************
* Description: Returns the files a user has purchased
*********************************************************/
function getUserFilePurchases(){
        return $this->userFilePurchases;
}


/*********************************************************
* Function: setUserFilePurchases(ArrayCollection $userFilePurchases)
*********************************************************
* Description: Sets the files that a user has purchased
*********************************************************/
function setUserFilePurchases(ArrayCollection $userFilePurchases){
        $this->userFilePurchases = $userFilePurchases;
}


/*********************************************************
* Function: addUserFilePurchases(File $file)
*********************************************************
* Description: Adds a file that a user has purchased
*********************************************************/
function addUserFilePurchases(File $file){
        $this->userFilePurchases->add($file);
}


/*********************************************************
* Function: removeUserFilePurchases(File $file)
*********************************************************
* Description: Removes a file that a user has purchased
*********************************************************/
function removeUserFilePurchases(File $file){
        $this->userFilePurchases->remove($file);
}
```

```
/**********************************************************
* Function: hasUserFilePurchases(File $file)
**********************************************************
* Description: Checks to see if a user has purchased a file
**********************************************************/
function hasUserFilePurchases(File $file){
        return $this->userFilePurchases->contains($file);
}


/**********************************************************
* Example Description: User purchases a file, then gets
* a refund. Get the ArrayCollection of purchased files by
* the user, then set the ArrayCollection of purchased files.
* Finally check if the user purchased the file
**********************************************************/
$file = new File;
$brokeBen = $thisUser;
$brokeBen->addUserFilePurchases($file);
$brokeBen->removeUserFilePurchase($file);

$benBought = $brokeBen->getUserFilePurchases();
$brokeBen->setUserFilePurchases($benBought);

$didBenBuy = $brokeBen->hasUserFilePurchases($file);
```

12.1.2.1.4.5 User Subscribed To

userSubscribedTo will store an ArrayCollection of all the subscriptions a user has made. userSubscribedTo has get, set, add, remove, and has functions.

Listing 30: User Subscribed To Array Collection

```
/**********************************************************
* Variable: userSubscribedTo
**********************************************************
* Description: Stores an ArrayCollection of subscriptions
* to other users made.
**********************************************************/
/**
* @OGM\ManyToMany(relation="subscribedTo")
*/
protected $userSubscribedTo;


/**********************************************************
```

```
* Function: getUserSubscribedTo()
*********************************************************
* Description: Returns the users that are subscribed to
*********************************************************/
function getUserSubscribedTo(){
        return $this->userSubscribedTo;
}


/*********************************************************
* Function: setUserSubscribedTo(ArrayCollection $userSubscribedTo)
*********************************************************
* Description: Sets the ArrayCollection of users subscribed
* to
*********************************************************/
function setUserSubscribedTo(ArrayCollection $userSubscribedTo){
        $this->userSubscribedTo = $userSubscribedTo;
}


/*********************************************************
* Function: addUserSubscribedTo(User $subscribeToUser)
*********************************************************
* Description: Adds a new user subscription
*********************************************************/
function addUserSubscribedTo(User $subscribeToUser){
        $this->userSubscribedTo->add($subscribeToUser);
}


/*********************************************************
* Function: removeUserSubscribedTo(User $subscribedToUser)
*********************************************************
* Description: Removes a user subscription
*********************************************************/
function removeUserSubscribedTo(User $subscribedToUser){
        $this->userSubscribedTo->removeElement($subscribedToUser);
}


/*********************************************************
* Function: hasUserSubscribedTo(User $subscribedToUser)
*********************************************************
* Description: Checks to see if the user made a subscription
* to the passed user.
*********************************************************/
```

```
function hasUserSubscribedTo(User $checkUser){
        return $this->userSubscribedTo->contains($checkUser);
}


/**********************************************************
* Example Description: Add two new user subscription, and
* then remove one of them. Get the users subscribed to and
* set the ArrayCollection. Then check if both the users
* made the subscription
**********************************************************/
$positivePatrice = $thisUser;
$negativeNancy = new User;
$subscribedToUser = new User;

$positivePatrice->addUserSubscribedTo($subscribedToUser);
$negativeNancy->addUserSubscribedTo($subscribedToUser);
$negativeNancy->removeUserSubscribedTo($subscribedToUser);

$nancySubscriptions = $negativeNancy->getUserSubscribedTo();
$patriceSubscription = $positivePatrice->getUserSubscribedTo();
```

12.1.2.1.4.6 User Subscribers

userSubscribers stores an ArrayCollection of users that subscribed to this user. userSubscribers has get, set, add, remove, and has function.

Listing 31: User Subscribers Array Collection

```
/**********************************************************
* Variable: userSubscribers
**********************************************************
* Description: userSubscribers
**********************************************************/
/**
* @OGM\ManyToMany(relation="subscribers")
*/
protected $userSubscribers;


/**********************************************************
* Function: getUserSubscribers()
**********************************************************
* Description: Returns the users that made a subscription
* to this user
**********************************************************/
function getUserSubscribers(){
```

```
        return $this->userSubscribers;
}


/*********************************************************
* Function: setUserSubscribers(ArrayCollection $userSubscribers)
*********************************************************
* Description: Sets the users that made a subscription to
* this user.
*********************************************************/
function setUserSubscribers(ArrayCollection $userSubscribers){
        $this->userSubscribers = $userSubscribers;
}


/*********************************************************
* Function: addUserSubscribers(User $subscriberUser)
*********************************************************
* Description: Adds a new subscriber to this user
*********************************************************/
function addUserSubscribers(User $subscribersUser){
        $this->userSubscribers->add($subscribersUser);
}


/*********************************************************
* Function: removeUserSubscribers(User $subscriberUser)
*********************************************************
* Description: Remove a subscriber from this user
*********************************************************/
function removeUserSubscribers(User $subscribersUser){
        $this->userSubscribers->removeElement($subscribersUser);
}


/*********************************************************
* Function: hasUserSubscribers(User $subscriberUser)
*********************************************************
* Description: Checks if a this user has a certain
* subscriber
*********************************************************/
function hasUserSubscribers(User $checkUser){
        return $this->userSubscribers->contains($checkUser);
}
```

```
/*******************************************************
* Example Description: Add a subscriber, get all the
* subscribers to this suer, set the subscribers to this user,
* and check if the user has that subscriber
*******************************************************/
$subscribedTo = $thisUser;
$newNelly = new User;
$subscribedTo->addUserSubscribers($newNelly);

$subscribers = $subscribedTo->getUserSubscribers();
$subscribedTo->setUserSubscribers($subscribers);

$hasSubscriber = $subscribedTo->hasUserSubscribers($newNelly);
```

12.1.2.1.4.7 User Owned Files

userOwnedFiles stores an ArrayCollection of file that the user owns. userOwnedFiles has get, set, add, remove, and has functions.

Listing 32: User Owned Files Array Collection

```
/*******************************************************
* Variable: userOwnedFiles
*******************************************************
* Description: An Array Collection of all the files owned by this
* user
*******************************************************/
/**
* @ODM\ManyToMany(relation="ownsFile")
*/
protected $userOwnedFiles;


/*******************************************************
* Function: getUserOwnedFiles()
*******************************************************
* Description: Returns the files that the user owns
*******************************************************/
function getUserOwnedFiles(){
        return $this->userOwnedFiles;
}


/*******************************************************
* Function: setUserOwnedFiles(ArrayCollection $userOwnedFiles)
*******************************************************
* Description: Sets the files the user owns.
```

```
*******************************************************/
function setUserOwnedFiles(ArrayCollection $userOwnedFiles){
        $this->userOwnedFiles = $userOwnedFiles;
}


/*******************************************************
* Function: addUserOwnedFiles(File $userOwnedFile)
*******************************************************
* Description: Adds a file that the user owns.
*******************************************************/
function addUserFileOwnedFiles(File $userOwnedFile){
        $this->userFileOwnedFiles->add($userOwnedFile);
}


/*******************************************************
* Function: removeUserOwnedFiles(File $userOwnedFile)
*******************************************************
* Description: Removes a file that the user owns.
*******************************************************/
function removeUserOwnedFiles(File $userOwnedFile){
        $this->userOwnedFiles->removeElement($userOwnedFile);
}


/*******************************************************
* Function: hasUserOwnedFiles(File $userOwnedFile)
*******************************************************
* Description: Checks to see if the user owns the file
*******************************************************/
function hasUserOwnedFiles(File $file){
        return $this->userOwnedFiles->contains($file);
}



/*******************************************************
* Example Description: User Adds a file, then removes it.
* Get the users owned files, and then sets it.
* Checks to see if the user owns the file.
*******************************************************/
$userOwned = $thisUser;
$file = new File;
$userOwned->addUserOwnedFiles($file);
$userOwned->removeUserOwnedFiles($file);
```

```
$userOwnedFile = $userOwned->getUserOwnedFiles();
$userOwned->setUserOwnedFiles($userOwnedFiles);

$doesUserOwn = $userOwned->hasUserOwnedFiles($file);
```

12.1.2.1.4.8 User Collaborated Files

userCollaboratedFiles stores an ArrayCollection of files collaborated in by the user. userCollaboratedFiles has get, set, add, remove, and has functions.

Listing 33: User Collaborated Files Array Collection

```
/**********************************************************
* Variable: userCollaboratedFiles
**********************************************************
* Description: Stores the files the user has collaborated in
**********************************************************/
protected $userCollaboratedFiles;


/**********************************************************
* Function: getUserCollaboratedFiles()
**********************************************************
* Description: Returns the files the user has collaborated
* in.
**********************************************************/
/**
* @OGM\ManyToMany(relation="collaboratedFile")
*/
function getUserCollaboratedFiles(){
        return $this->userCollaboratedFiles;
}


/**********************************************************
* Function: setUserCollaboratedFiles(ArrayCollection
    $userCollaboratedFile)
**********************************************************
* Description: Sets the files a user has collaborated in
**********************************************************/
function setUserCollaboratedFiles(ArrayCollection
    $userCollaboratedFiles){
        $this->userCollaboratedFiles = $userCollaboratedFiles;
}
```

```
/*********************************************************
* Function: addUserCollaboratedFiles(File $collaboratedFile)
*********************************************************
* Description: Adds a new file that the user has
* collaborated in.
*********************************************************/
function addUserCollaboratedFiles(File $collaboratedFile){
        $this->userCollaboratedFiles->add($collaboratedFile);
}


/*********************************************************
* Function: removeUserCollaboratedFiles(File $collaboratedFile)
*********************************************************
* Description: Removes a file a user collabored in.
*********************************************************/
function removeUserCollaboratedFiles(File $collaboratedFile){
        $this->userCollaboratedFiles->removeElement($collaboratedFile);
}


/*********************************************************
* Function: hasUserCollaboratedFiles(File $file)
*********************************************************
* Description: Checks if the user has collaborated in the
* file
*********************************************************/
function hasUserCollaboratedFiles(File $file){
        return $this->userCollaboratedFiles->contains($file);
}


/*********************************************************
* Example Description: Adds a new collaborated file, then
* immediately removes it. Get and then set the collaborated
* files for the user, and check if the user collaborated
* in the file
*********************************************************/
$lazyLarry = $thisUser;
$file = $thisFile;
$lazyLarry->addUserCollaboratedFiles($file);
$lazyLarry->removeUserCollaboratedFiles($file);

$lazyLarryCollaboration = $lazyLarry->getUserCollaboratedFiles();
$lazyLarry->setUserCollaboratedFiles($lazyLaryCollaboration();
```

```
$didLarryContribute = $lazyLarry->hasUserCollaboratedFiles($file);
```

12.1.2.1.4.9 User Profile Comments

userProfileComments stores an ArrayCollection of all the comments on a users profile page. userProfileComments has get, set, add, remove, and has functions:

Listing 34: User Profile Comments Array Collection

```
/**********************************************************
* Variable: userProfileComments
**********************************************************
* Description: Stores all the comments made on this
* users profile page
**********************************************************/
/**
* @ODM\ManyToMany(relation="profileComments")
*/
protected $userProfileComments;


/**********************************************************
* Function: getUserProfileComments
**********************************************************
* Description: Returns the comments made on this users
* profile page
**********************************************************/
function getUserProfileComments(){
        return $this->userProfileComments;
}


/**********************************************************
* Function: setUserProfileComments(ArrayCollection $userProfileComments
    )
**********************************************************
* Description: Sets the comments made on this users profile
* page.
**********************************************************/
function setUserProfileComments(ArrayCollection $userProfileComments){
        $this->userProfileComments = $userProfileComments;
}


/**********************************************************
* Function: addUserProfileComments(Comment $userComment)
**********************************************************
```

```
* Description: Adds a new comment on the users profile page
*********************************************************/
function addUserProfileComments(Comment $userComment){
        $this->userProfileComments->add($userComment);
}


/*********************************************************
* Function: removeUserProfileComments(Comment $userComment)
*********************************************************
* Description: Removes a comment on the users profile page
*********************************************************/
function removeUserProfileComments(Comment $userComment){
        $this->userProfileComments->removeElement($userComment);
}


/*********************************************************
* Function: hasUserProfileComments(Comment $userComment)
*********************************************************
* Description: Checks if the comment is on this users"
* profile page.
*********************************************************/
function hasUserProfileComments(Comment $userComment){
        return $this->userProfileComments->contains($userComment)
}


/*********************************************************
* Example Description: Add a comment, remove the comment,
* and then add a new comment. Get and set the user profile
* comments and then check which comments are on this users
* profile page.
*********************************************************/

$poorCommentText = 'lyk dis if u cri evertime';
$goodCommentText = 'This invokes copious amounts of despair on every
    occurence!';
$poorComment = new Comment;
$goodComment = new Comment;
$poorComment->setCommentText($poorCommentText);
$goodComment->setCommentText($goodCommentText);

$commentOn = $thisUser;
$commentOn->addUserProfileComments($poorComment);
$commentOn->addUserProfileComments($goodComment);
```

```
$commentOn->removeUserProfileComments($poorComment);

$profileComments = $commentOn->getUserProfileComments();
$commentOn->setUserProfileComments($profileComments);

$doesHaveGoodComment = $commentOn->hasUserProfileComments($goodComment)
    ;
$doesHavePoorComment = $commentOn->hasUserProfileComments($poorComment)
    ;
```

12.1.2.1.4.10 User Comments

userComments stores an ArrayCollection of the comments made by a user. userComments has get, set, add, remove, and has functions.

Listing 35: User Comments Array Collection

```
/**********************************************************
* Variable: userComments
**********************************************************
* Description: Stores an ArrayCollection of the comments
* made by this user
**********************************************************/
/**
* @OGM\ManyToMany(relation="userCommented")
*/
protected $userComments;


/**********************************************************
* Function: getUserComments()
**********************************************************
* Description: Returns the comments made by this user
**********************************************************/
function getUserComments(){
        return $this->userComments;
}


/**********************************************************
* Function: setUserComments(ArrayCollection $userComments)
**********************************************************
* Description: Sets the comments made by this user
**********************************************************/
function setUserComments(ArrayCollection $userComments){
        $this->userComments = $userComments;
}
```

```
/*********************************************************
* Function: addUserComments(Comment $userComment)
*********************************************************
* Description: Adds a new user comment.
*********************************************************/
function addUserComments(Comment $userComment){
        $this->userComments->add($userComment);
}


/*********************************************************
* Function: removeUserComments(Comment $userComment)
*********************************************************
* Description: Removes a user comment
*********************************************************/
function removeUserComments(Comment $userComment){
        $this->userComments->removeElement($userComment);
}


/*********************************************************
* Function: hasUserComments(Comment $userComment)
*********************************************************
* Description: Checks if the comment was made by this user
*********************************************************/
function hasUserComments(Comment $userComment){
        return $this->userComments->contains($userComment);
}


/*********************************************************
* Example Description: Add several comments, and remove a
* few of them. Then get and set the user comments and check
* which comments are still in the Array Collection
*********************************************************/
$commentGood1 = new Comment;
$commentGood2 = new Comment;
$commentGood3 = new Comment;
$commentBad1 = newComment;

$userWhoCommented = $thisUser;
$userWhoCommented->addUserComments($commentGood1);
$userWhoCommented->addUserComments($commentGood2);
```

```
$userWhoCommented->addUserComments($commentGood3);
$userWhoCommented->addUserComments($commentBad1);

$userWhoCommented->removeUserComments($commentGood2);
$userWhoCommented->removeUserComments($commentBad1);

$userComments = $userWhoCommented->getUserComments();
$userWhoCommented->setUserComments($userComments);

$madeGoodComment1 = $userWhoCommented->hasUserComments($commentGood1);
$madeGoodComment2 = $userWhoCommented->hasUserComments($commentGood2);
$madeGoodComment3 = $userWhoCommented->hasUserComments($commentGood3);
$madeBadComment1 = $userWhoCommented->hasUserComments($commentBad1);
```

12.1.2.1.5 User Class Construction

This section will breifly go over the construction of the user entity.

12.1.2.1.5.1 User Construct

Listing 36: User Construct Function

```
function __construct(){
  /**********************************************************
  * userFileX Description: An array collection corresponding
  * to how the user interacted with a file
  **********************************************************/
      $this->userFileLikes = new ArrayCollection;
      $this->userFileViews = new ArrayCollection;
      $this->userFileDislikes = new ArrayCollection;
      $this->userFilePurchases = new ArrayCollection;


  /**********************************************************
  * userSubscribeX Description: An array collection
  * corresponding to how the users subscribers and subscriptions
  **********************************************************/
        $this->userSubscribedTo = new ArrayCollection;
        $this->userSubscribers = new ArrayCollection;


  /**********************************************************
  * userOwned/Collaborated Description: An array collection
  * correspond to files the user owns and collaborated files
  * they themselves have worked on.
  **********************************************************/
```

```
        $this->userOwnedFiles = new ArrayCollection;
        $this->userCollaboratedFiles = new ArrayCollection;


    /**********************************************************
    * user(Profile)Comments Description: An array collection
    * correspond to comments a user has made, or the comments
    * on their user page.
    **********************************************************/
        $this->userProfileComments = new ArrayCollection;
        $this->userComments = new ArrayCollection;


    /**********************************************************
    * userUniqueId Description: Dynamically creates a unique
    * id for the user node.
    **********************************************************/
        $this->userUniqueId = uniqid();
}

    /**********************************************************
    * How To Create A New Instance of a User Class.
    **********************************************************/
$registerThisUser = new User;
```

12.1.2.2 File Entity

The File Entity Class has properites in order to track the popularity of files, which users interacted with the file, and other file information

12.1.2.2.1 Indexes

File Indexes are used in searching and finding Files relatively quickly.

12.1.2.2.1.1 File Id

fileId will store the Id that is generated with the OGM Auto index. file has get and set functions.

Listing 37: File Id Index

```
/**********************************************************
* Variable: fileId
**********************************************************
* Description: Stores the files id
**********************************************************/
/**
* @OGM\Auto
*/
```

```
protected $fileId;


/*********************************************************
* Function: getFileid()
*********************************************************
* Description: Returns the files id
*********************************************************/
function getFileId(){
        return $this->fileId;
}


/*********************************************************
* Function: setFileId($fileId)
*********************************************************
* Description: Sets the files id
*********************************************************/
function setFileId($fileId){
        $this->fileId = $fileId;
}

/*********************************************************
* Example Description: Get the current file id and then set
* the current file id
*********************************************************/
$thisFileId = $thisFile->getFileId();
$thisFile->setFileId($fileId);
```

12.1.2.2.1.2 File Unique Id

fileUniqueId will store a uniquely generated id. fileUniqueId has a get function.

Listing 38: File Unique Id Index

```
/*********************************************************
* Variable: fileUniqueId
*********************************************************
* Description: Stores the files unique id
*********************************************************/
/**
* @OGM\Property
* @OGM\Index
*/
protected $fileUniqueId;

/*********************************************************
```

```
* Function: getFileUniqueId()
*********************************************************
* Description: Returns the files unique id
*********************************************************/
function getFileUniqueId(){
        return $this->fileUniqueId;
}


/*********************************************************
* Example Description:  Get the current users unique id
*********************************************************/
$thisFileUniqueId = $thisFile->getFileUniqueId();
```

12.1.2.2.1.3 File Name

fileName will store the files name. fileName has gen and set functions.

Listing 39: File Name Index

```
/*********************************************************
* Variable: fileName
*********************************************************
* Description: Stores the files name
*********************************************************/
/**
* @OGM\Property
* @OGM\Index
*/
protected $fileName;


/*********************************************************
* Function: getFileName()
*********************************************************
* Description: Returns the files name
*********************************************************/
function getFileName(){
                return $this->fileName;
        }


/*********************************************************
* Function: setFileName($fileName)
*********************************************************
* Description: Sets the files name
*********************************************************/
```

```
function setFileName($fileName){
        $this->fileName = $fileName;
}

/*********************************************************
* Example Description: Get the files current name and set
* a new file name
*********************************************************/
$currentFileName = $thisFile->getFileName();
$newName = 'Zoboomafoo';
$thisFile->setFileName($newName);
```

12.1.2.2.2 Properties

File Properties consist of the basic file information. That is, the number of comments the file has, the number of views the file has, the number of likes the file has, etc.

12.1.2.2.2.1 File Type

fileType will store the type of file, whether it be jpg or mp4. fileType has get and set functions.

Listing 40: File Type Property

```
/*********************************************************
* Variable: fileType
*********************************************************
* Description: Stores the file type
*********************************************************/
/**
* @OGM\Property
*/
protected $fileType;


/*********************************************************
* Function: getFileType()
*********************************************************
* Description: Returns the file type
*********************************************************/
function getFileType(){
        return $this->fileType;
}


/*********************************************************
* Function: setFileType($fileType)
*********************************************************
* Description: Sets the file type
```

```
*********************************************************/
function setFileType($fileType){
        $this->fileType = $fileType;
}

/*********************************************************
* Example Description: Get the current file type, and if it
* is jpg set the new file type to png.
*********************************************************/
$fileType = $thisFile->getFileType();
if($fileType == 'jpg'){
        $thisFile->setFileType('png');
}
```

12.1.2.2.2.2 File Views

fileViews will store the total views the file has. fileViews has get and set functions.

Listing 41: File Views Property

```
/*********************************************************
* Variable: fileViews
*********************************************************
* Description: Stores the total views for the file
*********************************************************/
/**
* @OGM\Property
*/
protected $fileViews;


/*********************************************************
* Function: getFileViews()
*********************************************************
* Description: Returns the total views for the file
*********************************************************/
function getFileViews(){
        return $this->fileViews;
}


/*********************************************************
* Function: setFileViews($fileViews)
*********************************************************
* Description: Sets the total views for the file
*********************************************************/
function setFileViews($fileViews){
```

```
        $this->fileViews = $fileViews;
}

/*************************************************************
* Example Description: If the current file views are 0, set
* the new file views to be 100.
*************************************************************/
$fileViews = $thisFile->getFileViews();
if($fileViews==0){
        $newFileViews = 100;
        $thisFile->setFileViews($newFileViews);
}
```

12.1.2.2.2.3 File Likes

fileLikes will store the total likes for the file. fileLikes has get and set functions.

Listing 42: File Likes Property

```
/*************************************************************
* Variable: fileLikes
*************************************************************
* Description: Stores the total number of file likes
*************************************************************/
/**
* @ODM\Property
*/
protected $fileLikes;


/*************************************************************
* Function: getFileLikes()
*************************************************************
* Description: Returns the total number of file likes
*************************************************************/
function getFileLikes(){
        return $this->fileLikes;
}


/*************************************************************
* Function: setFileLikes($fileLikes)
*************************************************************
* Description: Sets the total number of file likes
*************************************************************/
function setFileLikes($fileLikes){
        $this->fileLikes = $fileLikes;
```

```
}


/*********************************************************
* Example Description: Get the total file likes and set
* the new total file likes after a file like.
*********************************************************/
$fileLikes = $thisFile->getFileLikes();
$newFileLikes = $fileLikes + 1;
$thisFile->setFileLikes($newFileLikes);
```

12.1.2.2.2.4 File Dislikes

fileDislikes will store the total number of dislikes for a file. fileDislikes has get and set functions.

Listing 43: File Dislikes Property

```
/*********************************************************
* Variable: fileDislikes
*********************************************************
* Description: Stores the total number of file dislikes
*********************************************************/
/**
* @OGM\Property
*/
protected $fileDislikes;


/*********************************************************
* Function: getFileDislikes()
*********************************************************
* Description: Returns the total number of file dislikes
*********************************************************/
function getFileDislikes(){
        return $this->fileDislikes;
}


/*********************************************************
* Function: setFileDislikes($fileDislikes)
*********************************************************
* Description: Sets the total number of file dislikes
*********************************************************/
function setFileDislikes($fileDislikes){
        $this->fileDislikes = $fileDislikes;
}
```

```
/*********************************************************
* Example Description: If the user has any file dislikes,
* set the new file dislikes after a user undislikes a file
*********************************************************/
$fileDislikes = $thisFile->getFileDislikes();
if($fileDislikes>0){
        $newFileDislikes = $fileDislikes - 1;
    $thisFile->setFileDislikes($newFileDislikes);
}
```

12.1.2.2.2.5 File Upload Date

fileUploadDate will store when the file was update as a time stamp. fileUploadDate has get and set functions.

Listing 44: File Upload Date Property

```
/*********************************************************
* Variable: fileUploadDate
*********************************************************
* Description: Stores when the file was uploaded
*********************************************************/
/**
* @OGM\Property(format="date")
*/
protected $fileUploadDate;


/*********************************************************
* Function: getFileUploadDate()
*********************************************************
* Description: Returns when the file was uploaded
*********************************************************/
function getFileUploadDate(){
        return $this->fileUploadDate;
}


/*********************************************************
* Function: setFileUploadDate($fileUploadDate)
*********************************************************
* Description: Sets when the file was uploaded
*********************************************************/
function setFileUploadDate($fileUploadDate){
        $this->fileUploadDate = $fileUploadDate;
}
```

```
/*********************************************************
* Example Description: Get the file upload date, and set a
* new files upload date.
*********************************************************/
$fileUploadDate = $thisFile->getFileUploadDate();
$newFile = new File;
$fileUploadDate = date("Y-m-d h:i:s");
$newFile->setFileUploadDate($fileUploadDate);
```

12.1.2.2.2.6 File Description

fileDescription will store a short description for the file. fileDescription has get and set functions.

Listing 45: File Description Property

```
/*********************************************************
* Variable: fileDescr
*********************************************************
* Description: Stores the file description
*********************************************************/
/**
* @ORM\Property
*/
protected $fileDescr;


/*********************************************************
* Function: getFileDescr()
*********************************************************
* Description: Returns the file description
*********************************************************/
function getFileDescr(){
        return $this->fileDescr;
}


/*********************************************************
* Function: setFileDescr($fileDescr)
*********************************************************
* Description: Sets the file Descr
*********************************************************/
function setFileDescr($fileDescr){
        $this->fileDescr = $fileDescr;
}
```

```
/*******************************************************
* Example Description: Get the file description, and set
* the new file description based on the old description
*******************************************************/
$oldFileDescr = $thisFile->getFileDescr();
if($oldFileDescr == 'Awesomely Possum'){
        $newFileDescr = 'Definitely Awesomely Possum';
    $thisFile->setFileDescr($newFileDescr);
}
else{
        $newFileDescr = 'Not Awesomely Possum';
    $thisFile->setFileDescr($newFileDescr);
}
```

12.1.2.2.2.7 File Explicit

fileExplicit will store if the file is explicit or not. fileExplicit has get and set functions.

Listing 46: File Explicit Property

```
/*******************************************************
* Variable: fileExplicit
*******************************************************
* Description: Stores if the file is explicit
*******************************************************/
/**
* @OGM\Property
*/
protected $fileExplicit


/*******************************************************
* Function: getFileExplicit()
*******************************************************
* Description: Returns if the file is explicit
*******************************************************/
public function getFileExplicit(){
        return $this->fileExplicit;
}


/*******************************************************
* Function: setFileExplicit($fileExplicit)
*******************************************************
* Description: Sets if the file is explicit
*******************************************************/
```

```
/*********************************************************
* Example Description: Get if the file is explicit, if it is
* not, make it explicit.
*********************************************************/
$isFileExplicit = $thisFile->getFileExplicit();
if(!$isFileExplicit){
        $thisFile->setFileExplicit(true);
}
```

12.1.2.2.2.8 File Price

filePrice will store what price the user is selling the file for. Zero indicates the file is free. filePrice has get and set functions.

Listing 47: File Price Property

```
/*********************************************************
* Variable: filePrice
*********************************************************
* Description: Stores the files price
*********************************************************/
/**
* @OGM\Property
*/
protected $filePrice;


/*********************************************************
* Function: getFilePrice()
*********************************************************
* Description: Returns the file price
*********************************************************/
function getFilePrice(){
        return $this->filePrice;
}


/*********************************************************
* Function: setFilePrice($filePrice)
*********************************************************
* Description: Sets the files price
*********************************************************/
function setFilePrice($filePrice){
        $this->filePrice = $filePrice;
}
```

```
/*********************************************************
* Example Description: Get the files price. If it is not
* free, set the file price to free.
*********************************************************/
$filePrice = $thisFile->getFilePrice();
if($filePrice>0){
        $newFilePrice=0;
    $thisFile->setFilePrice($newFilePrice);
}
```

12.1.2.2.2.9 File Can Edit Price

fileCanEditPrice is an admistrative property. If the user keeps changing the price too frequently, we may be forced to set a price. fileCanEditPrice has get and set functions.

Listing 48: File Can Edit Price Property

```
/*********************************************************
* Variable: fileCanEditPrice
*********************************************************
* Description: Stores if the files price can be edited
*********************************************************/
/**
* @OGM\Property
*/
protected $fileCanEditPrice

/*********************************************************
* Function: getFileCanEditPrice()
*********************************************************
* Description: Returns if the files price can be edited
*********************************************************/
function getFileCanEditPrice(){
        return $this->fileCanEditPrice;
}


/*********************************************************
* Function: setFileCanEditPrice($fileCanEditPrice)
*********************************************************
* Description: Sets if the files price can be edited
*********************************************************/
function setFileCanEditPrice($fileCanEditPrice){
        $this->fileCanEditPrice = $fileCanEditPrice;
}
```

```
/*********************************************************
* Example Description: If the current files price can't be
* edited, set the file so the price can be edited.
*********************************************************/
$fileCanEditPrice = $thisFile->getFileCanEditPrice();
if(!$fileCanEditPrice){
        $thisFile->setFileCanEditPrice(true);
}
```

12.1.2.2.2.10 File Owner

fileOwner stores a user node that uses the OGM Many To One Property. fileOwner has get and set functions.

Listing 49: File Owner Property

```
/*********************************************************
* Variable: fileOwner
*********************************************************
* Description: Stores the file owner
*********************************************************/
/**
* @OGM\ManyToOne(relation="ownsFile")
*/
protected $fileOwner


/*********************************************************
* Function: getFileOwner()
*********************************************************
* Description: Returns the files owner
*********************************************************/
function getFileOwner(){
        return $this->fileOwner;
}


/*********************************************************
* Function: setFileOwner($fileOwner)
*********************************************************
* Description: Sets the file owner
*********************************************************/
function setFileOwner($fileOwner){
        $this->fileOwner = $fileOwner;
}

/*********************************************************
```

```
* Example Description: Get the current file owner and if their
* name isn't Bob, set the file owner to bob.
*****************************************************/
$isBob = $thisUser->getUserName();
$bob = new User;
$fileOwner = $thisFile->getFileOwner();
if($isBob != $fileOwner){
        $thisFile->setFileOwner($bob);
}
else{
        $thisFile->setFileOwner($thisUser);
}
```

12.1.2.2.2.11 File Total Comments

fileTotalComments stores the total number of comments on the file. fileTotalComments has get and set functions.

Listing 50: File Total Comments Property

```
/*******************************************************
* Variable: fileTotalComments
*******************************************************
* Description: Stores the total number of file comments
*******************************************************/
/**
* @OGM\Property
*/
protected $fileTotalComments;


/*******************************************************
* Function: getFileTotalComments()
*******************************************************
* Description: Returns the total number of file comments
*******************************************************/
function getFileTotalComments(){
        return $this->fileTotalComments;
}


/*******************************************************
* Function: setFileTotalComments($fileTotalComments)
*******************************************************
* Description: Sets the total number of file comments
*******************************************************/
function setFileTotalComments($fileTotalComments){
```

```
        $this->fileTotalComments = $fileTotalComments;
}


/*********************************************************
* Example Description: Gets the current file comments, and
* sets the new file comments after a new comment.
*********************************************************/
$fileCommentCount = $thisFile->getFileTotalComments();
```

12.1.2.2.2.12 File Preview

filePreview will store the name of the file preview. filePreview has get and set functions

Listing 51: File Preview Property

```
/*********************************************************
* Variable: filePreview
*********************************************************
* Description: Stores the file preview name
*********************************************************/
/**
* @ODM\Property
*/
protected $filePreview


/*********************************************************
* Function: getFilePreview()
*********************************************************
* Description: Returns the file preview.
*********************************************************/
function getFilePreview(){
        return $this->filePreview;
}


/*********************************************************
* Function: setFilePreview($filePreview)
*********************************************************
* Description: Sets the file preview name
*********************************************************/
function setFilePreview($filePreview){
        $this->filePreview = $filePreview;
}
```

```
/*********************************************************
* Example Description: Get file file preview name and get
* its directory path. Then set a new file name.
*********************************************************/
$filePreview = $thisFile->getFilePreview();
$filePreviewDir = '/images/'.$thisFile->getFileId().'/'.$filePreview./
$newFilePreview = 'Preview.jpg';
$thisFile->setFilePreview($newFilePreview);
```

12.1.2.2.2.13 File Is Paid

fileIsPaid is just a quick way to look up if the file is free or not. fileIsPaid has get and set functions.

Listing 52: File Is Paid Property

```
/*********************************************************
* Variable: fileIsPaid
*********************************************************
* Description: Stores if the file is paid or not.
*********************************************************/
/**
* @OGM\Property
*/
protected $fileIsPaid;


/*********************************************************
* Function: getFileIsPaid()
*********************************************************
* Description: Returns if the file is free or not
*********************************************************/
function getFileIsPaid(){
        return $this->fileIsPaid;
}


/*********************************************************
* Function: setFileFileIsPaid($fileIsPaid)
*********************************************************
* Description: Sets if the file is free or not
*********************************************************/
function setFileIsPaid($fileIsPaid){
        $this->fileIsPaid = $fileIsPaid;
}


/*********************************************************
```

```
* Example Description: Get if the current file is paid, and
* if its not free, make it free.
*****************************************************/
$fileIsPaid = $thisFile->getFileIsPaid();
if($fileIsPaid){
        $thisFile->setFileIsPaid(false);
}
```

12.1.2.2.3 Arrays

Arrays have similar to Properties but they have an additional add fucntion. Arrays used by files be File Tags and File Categories.

12.1.2.2.3.1 File Tags

fileTags will store the tags associated with the file in an array. fileTags has get, set, and add functions.

Listing 53: File Tags Array

```
/*********************************************************
* Variable: fileTags
*********************************************************
* Description: Stores the file tags of the file in an
* array.
*********************************************************/
/**
* @OGM\Property(format="array")
*/
protected $fileTags;


/*********************************************************
* Function: getFileTags()
*********************************************************
* Description: Returns the array of file tags.
*********************************************************/
public function getFileTags(){
        return $this->fileTags;
}


/*********************************************************
* Function: setFileTags(array $fileTags)
*********************************************************
* Description: Sets the array of file tags
*********************************************************/
public function setFileCategories(array $fileCategories){
```

```
        $this->fileCategories = array();
        foreach($fileCategories as $fileCategory){
                $this->addFileCategories($fileCategory);
        }
}


/*****************************************************
* Function: addFileTags($fileTag)
*****************************************************
* Description: Adds a new file tag.
*****************************************************/
public function addFileCategories($fileCategory){
        $this->fileCategories[] = $fileCategory;
}


/*****************************************************
* Example Description: Get the current file tags. Then set
* the file tags to the array value.
*****************************************************/
$fileTags = $thisFile->getFileTags();
$thisFile->setFileTags($thisFile);
```

12.1.2.2.3.2 File Categories

fileCategories will store the files categories. fileCategories has get, set, and add fucntions

Listing 54: File Categories Array

```
/*****************************************************
* Variable: fileCategories
*****************************************************
* Description: Stores all the comments made on this
* users profile page
*****************************************************/
/**
* @OGM\Property(format="array")
*/
protected $fileCategories


/*****************************************************
* Function: getFileCategories()
*****************************************************
* Description: Returns the array of file categories
*****************************************************/
```

```
/*******************************************************
* Function:  setFileCategories(array $fileCategories)
*******************************************************
* Description: Sets the array of file categories
*******************************************************/
public function setFileCategories(array $fileCategories){
        $this->fileCategories = array();
        foreach($fileCategories as $fileCategory){
                $this->addFileCategories($fileCategory);
        }
}


/*******************************************************
* Function: addFileCategories($fileCategory)
*******************************************************
* Description: Adds a new file category to the array.
* User in the setFileCategories function.
*******************************************************/
public function addFileCategories($fileCategory){
        $this->fileCategories[] = $fileCategory;
}


/*******************************************************
* Example Description: Get the current file categories,
* and set the new file categories to the array provided
*******************************************************/
$fileCategories->$thisFile->getFileCategories();
$newFileCategories = array("Humor", "Drama", "Action");
$thisFile->setFileCategories($newFileCategories);
```

12.1.2.2.4 ArrayCollections

File ArrayCollections hold collections of nodes relevant to the file. These ArrayCollection can hold information such as which users like the file, which user commented on the file, and what users collaborated in the file.

12.1.2.2.4.1 File User Views

fileUserViews will store an ArrayCollection of users who viewed the file. fileUserViews has get, set, add, remove, and has functions.

Listing 55: File User Views ArrayCollection

```
/**********************************************************
* Variable: fileUserViews
**********************************************************
* Description: Stores the users who viewed the file
**********************************************************/
/**
* @ODM\ManyToMany(relation="userViewedFile")
*/
protected $fileUserViews;


/**********************************************************
* Function: getFileUserViews()
**********************************************************
* Description: Returns the Users who viewed the file
**********************************************************/
function getFileUserViews(){
        return $this->fileUserViews;
}


/**********************************************************
* Function: setFileUserViews(ArrayCollection $fileUserViews)
**********************************************************
* Description: Sets the users that viewed the file
**********************************************************/
function setFileUserViews(ArrayCollection $fileUserViews){
        $this->fileUserViews = $fileUserViews;
}


/**********************************************************
* Function: addFileUserViews(User $userViewed)
**********************************************************
* Description: Adds a new user that viewed the file
**********************************************************/
function addFileUserViews(User $userViewed){
        $this->fileUserViews->add($userViewed);
}


/**********************************************************
* Function: removeFileUserViews(User $userViewed)
**********************************************************
* Description: Removes a user that viewed the file
**********************************************************/
```

```
function removeFileUserViews(User $userViewed){
        $this->fileUserViews->removeElement($userViewed);
}


/**********************************************************
* Function: hasFileUserViews(User $user)
**********************************************************
* Description: Checks to see if the user viewed the file
**********************************************************/
function hasFileUserViews(User $user){
        return $this->fileUserViews->contains($user);
}

/**********************************************************
* Example Description: Adds a user file view, then removes
* it. Gets the users that viewed the files, then sets it.
* Finally we check to see if the user viewed the file
**********************************************************/
$fileViewed = $thisFile;
$userViewed = $thisUser;
$fileViewed->addFileUserViews($userViewed);
$fileViewed->removeFileUserViews($userViewed);

$usersThatViewedFile = $fileViews->getFileUserViews();
$fileViewed->setFileUserViews($usersThatViewedFile);

$hasUserViewed = $fileViewed->hasFileUserViews($userViewed);
```

12.1.2.2.4.2 File User Likes

fileUserLikes will store an Array Collection of all the users that liked the file. fileUserLikes has get, set, add, remove, and has functions.

Listing 56: File User Likes ArrayCollection

```
/**********************************************************
* Variable: fileUserLikes
**********************************************************
* Description: Stores all the users that liked the file
**********************************************************/
/**
* @OGM\ManyToMany(relation="userLikedFiles")
*/
protected $fileUserLikes;
```

```
/**********************************************************
* Function: getFileUserLikes()
**********************************************************
* Description: Returns the users that liked this file
**********************************************************/
function getFileUserLikes(){
        return $this->fileUserLikes;
}


/**********************************************************
* Function: setFileUserLikes(ArrayCollection $fileUserLikes)
**********************************************************
* Description: Sets the users that liked this file
**********************************************************/
function setFileUserLikes(ArrayCollection $fileUserLikes){
        $this->fileUserLikes = $fileUserLikes;
}


/**********************************************************
* Function: addFileUserLikes(User $userLiked)
**********************************************************
* Description: Adds a new user that liked the file
**********************************************************/
function addFileUserLikes(User $userLiked){
        $this->fileUserLikes->add($userLiked;
}


/**********************************************************
* Function: removeFileUserLikes(User $userLiked)
**********************************************************
* Description: Removes a user that liked the file
**********************************************************/
function removeFileUserLikes(User $userLiked){
        $this->fileUserLikes->removeElement($userLiked);
}


/**********************************************************
* Function: hasFileUserLikes(User $user)
**********************************************************
* Description: Checks to see if the user has liked the file
**********************************************************/
function hasFileUserLikes(User $user){
```

```
        return $this->fileUserLikes->contains($user);
}

/**********************************************************
* Example Description: Remove a user file like, then add
* that file like from that same user. Get and Set the users
* that liked the file, then check if the user liked the file
**********************************************************/
$unsureSusan = $thisUser;
$fileLiked = $thisFile;
$fileLiked->removeFileUserLikes($unsureSusan);
$fileLiked->addFileUserLikes($unsureSusan);

$usersThatLikedFile = $fileLiked->getFileUserLikes();
$fileLiked->setFileUserLikes($usersThatLikedFile);

$didSusanLike = $fileLiked->hasFileUserLikes($unsureSusan);
```

12.1.2.2.4.3 File User Dislikes

filsUserDislikes will store an ArrayCollection of users that disliked the file. fileUserDislikes has get, set, add, remove, and has functions.

Listing 57: File User Dislikes ArrayCollection

```
/**********************************************************
* Variable: fileUserDislikes
**********************************************************
* Description: Stores the users that disliked the file
**********************************************************/
/**
* @OGM\ManyToMany(relation="userDislikesFile")
*/
protected $fileUserDisliked


/**********************************************************
* Function: getFileUserDislikes()
**********************************************************
* Description: Returns the users that disliked the file
**********************************************************/
function getFileUserDislikes(){
        return $this->fileComments;
        }

/**********************************************************
* Function: setFileUserDislikes(ArrayCollection $fileUserDislikes)
```

```
*********************************************************
* Description: Sets the users that disliked the file
*********************************************************/
function setFileUserDislikes(ArrayCollection $fileUserDislikes){
        $this->fileUserDislikes = $fileUserDislikes;
}


/*********************************************************
* Function: addFileUserDislikes(User $userDislike)
*********************************************************
* Description: Adds a new user that disliked the file
*********************************************************/
function addFileUserDislikes(User $userDislike){
        $this->fileUserDislikes->add($userDislike);
}


/*********************************************************
* Function: removeFileUserDislikes(User $userDislike)
*********************************************************
* Description: Removes a user that disliked the file
*********************************************************/
function removeUserDislikes(User $userDislike){
        $this->fileUserDislikes->removeElement($userDislike);
}


/*********************************************************
* Function: hasFileUserDislikes(User $user)
*********************************************************
* Description: Checks to see if the user disliked the file
*********************************************************/
function hasFileUserDislikes(User $user){
        return $this->fileUserDislikes->contains($user);
}

/*********************************************************
* Example Description: Add a user that disliked the file,
* then remove that user dislike. Get and set the user
* disliked of the file and check if the user disliked the
* file.
*********************************************************/
$forgivingFred = $thisUser;
$file = $thisFile;
$thisFile->addFileUserDislikes($forgivingFred);
```

```
$thisFile->removeFileUserDislikes($forgivingFred);

$usersWhoDisliked = $thisFile->getFileUserDislikes();
$thisFile->setFileUserDislikes($usersWhoDisliked);

$didFredDisliked = $thisFile->hasFileUserDislikes($forgivingFred);
```

12.1.2.2.4.4 File User Purchases

fileUserPurchases will store all the users that bought this file. fileUserPurchases has get, set, add, remove, and has functions.

Listing 58: File User Purchases ArrayCollection

```
/**********************************************************
* Variable: fileUserPurchases
**********************************************************
* Description: Stores the ArrayCollection of users that
* purchased this file
**********************************************************/
protected $fileUserPurchases;


/**********************************************************
* Function: getFileUserPurchases()
**********************************************************
* Description: Returns the users that purchased this file
**********************************************************/
/**
* @ODM\ManyToMany(relation="userPurchasesFile")
*/
function getFileUserPurchases(){
        return $this->fileUserPurchases;
}


/**********************************************************
* Function: setFileUserPurchases(ArrayCollection $fileUserPurchases)
**********************************************************
* Description: Sets the users that purchased this file
**********************************************************/
function setFileUserPurchases(ArrayCollection $fileUserPurchases){
        $this->fileUserPurchases = $fileUserPurchases;
}


/**********************************************************
```

```
* Function: addFileUserPurchases(User $userPurchase)
*********************************************************
* Description: Adds a new user that purchased this file
*********************************************************/
function addFileUserPurchases(User $userPurchase){
        $this->fileUserPurchases->add($userPurchase);
}


/*********************************************************
* Function: removeFileUserPurchases(User $userPurchase)
*********************************************************
* Description: Removes a user that purchased the file
*********************************************************/
function removeFileUserPurchases(User $userPurchase){
        $this->fileUserPurchases->removeElement($userPurchase);
}


/*********************************************************
* Function: hasFileUserPurchases(User $user)
*********************************************************
* Description: Checks to see if
*********************************************************/
function hasFileUserPurchases(User $user){
        return $this->fileUserPurchases->contains($user);
}


/*********************************************************
* Example Description: Adds a user that purchased the file,
* removes them, and then adds them again. Gets and sets the
* users that purchased the file, and then checks if the
* user purchased the file
*********************************************************/
$bankruptBill = $thisUser;
$file = $thisFile;
$file->addFileUserPurchases($bankruptBill);
$file->removeFileUserPurchases($bankruptBill);
$file->addFileUserPurchases($bankruptBill);

$usersWhoPurchased = $file->getFileUserPurchases();
$file->setFileUserPurchases($usersWhoPurchased);

$didBillBuy = $file->hasFileUserPurchases($bankruptBill);
```

12.1.2.2.4.5 File Collaborators

fileCollaborators will store the users that collaborated in this file. fileCollaboraters has get, set, add, remove, and has functions.

Listing 59: File Collaborators ArrayCollection

```
/**********************************************************
* Variable: fileCollaborators
**********************************************************
* Description: Stores the users that collaborated in the
* file
**********************************************************/
/**
* @OGM\ManyToMany(relation="collaboratedFile")
*/
protected $fileCollaboraters;


/**********************************************************
* Function: getFileCollaboraters()
**********************************************************
* Description: Returns the users that collaborated in the
* file
**********************************************************/
function getFileCollaborators(){
        return $this->fileCollaborators;
}


/**********************************************************
* Function: setFileCollaboraters(ArrayCollection $fileCollaborators)
**********************************************************
* Description: Sets the users that collaborated in the file
**********************************************************/
function setFileCollaborators(ArrayCollection $fileCollaborators){
        $this->fileCollaborators = $fileCollaborators;
        }


/**********************************************************
* Function: addFileCollaborators(User $user)
**********************************************************
* Description: Adds a new file collaborator
**********************************************************/
function addFileCollaborator(User $user){
        $this->fileCollaborators->add($user);
}
```

```
/*********************************************************
* Function: removeFileCollaborators(User $user)
*********************************************************
* Description: Removes a file collaborator
*********************************************************/
function removeFileCollaborator(User $user){
        $this->fileCollaborators->removeElement($user);
}


/*********************************************************
* Function: hasFileCollaborators(User $user)
*********************************************************
* Description: Checks to see if the user collaborated in
* the file.
*********************************************************/
function hasFileCollaborators(User $user){
        return $this->fileCollaborators->contains($user);
}


/*********************************************************
* Example Description: Add two new file collaborators, and
* then remove one. Get and set the file collaborators and
* check which users collaborated in the files.
*********************************************************/
$awfulAidan = new User;
$busyBen = $thisUser;
$file = $thisFile;

$file->addFileCollaborators($awfulAidan);
$file->addFileCollaborators($busyBen);
$file->removeFileCollaborators($awfulAidan);

$usersWhoCollaborated = $file->getFileCollaborators();
$file->setFileCollaborators($usersWhoCollaborated);

$didAidanContribute = $file->hasFileCollaborators($awfulAidan);
$didBenContribute = $file->hasFileCollaborators($busyBen);
```

12.1.2.2.4.6 File Comments

fileComments will store an ArrayCollection of all the comments on this file. fileComments has get, set, add, remove, and has functions.

Listing 60: File Comments ArrayCollection

```
/**********************************************************
* Variable: fileComments
**********************************************************
* Description: Stores the comments made on a file
**********************************************************/
/**
* @ODM\ManyToMany(relation="comments")
*/
protected $fileComments;


/**********************************************************
* Function: getFileFileComments()
**********************************************************
* Description: Returns the comments made on this file page
**********************************************************/
function getFileComments(){
        return $this->fileComments;
}


/**********************************************************
* Function: setFileComments(ArrayCollection $fileComments)
**********************************************************
* Description: Sets the comments made on the file page
**********************************************************/
function setFileComments(ArrayCollection $fileComments){
        $this->fileComments = $fileComments;
}


/**********************************************************
* Function: addFileComments(Comment $comment)
**********************************************************
* Description: Adds a new file comment
**********************************************************/
function addFileComments(Comment $comment){
        $this->fileComments->add($comment);
}


/**********************************************************
* Function: removeFileComments(Comment $comment)
**********************************************************
* Description: Removes a file comment
```

```
*******************************************************/
function removeFileComments(Comment $comment){
        $this->fileComments->removeElement($comment);
}


/*******************************************************
* Function: hasFileComments(Comment $comment)
*******************************************************
* Description: Checks to see if the file has that comment
*******************************************************/
function hasFileComments(Comment $comment){
        return $this->fileComments->contains($comment);
}


/*******************************************************
* Example Description: Add a file comment, then remove it.
* Get and set the file comments, and then check if that
* comment was made
*******************************************************/
$awfulFileComment = new Comment;
$file = $thisFile;

$file->addFileComments($awfulFileComment);
$file->removeFileComments($awfulFileComment);

$commentsOnFile = $file->getFileComments();
$file->setFileComments($commentsOnFile);

$doesFileHaveAwfulComment = $file->hasFileComments($awfulFileComment);
```

12.1.2.2.5 File Construction

This section will breifly go over the construction of the file entity.

12.1.2.2.5.1 File Construct

Listing 61: File Construct Function

```
function __construct(){
        /*******************************************************
    * Arrays: In the construct function, the arrays are auto-
    * generated as empty arrays to be filled.
    *******************************************************/
```

```
        $this->fileTags = array();
        $this->fileCategories = array();

        /**********************************************************
    * ArrayCollection: In the construct function array collections
    * are autogenerated as empty array collections. Their uses
    * are explained above.
    **********************************************************/
        $this->fileCollaborators = new ArrayCollection;
        $this->fileComments = new ArrayCollection;

        $this->fileUserViews = new ArrayCollection;
        $this->fileUserLikes = new ArrayCollection;
        $this->fileUseraDislikes = new ArrayCollection;
        $this->fileUserPurchases = new ArrayCollection;


    /**********************************************************
    * UniqueId: Auto-generates the unique id required for the
    * file class. This is used in indexing.
    **********************************************************/
        $this->fileUniqueId = uniqid();
}

/**********************************************************
* How to Create a New File Entity
**********************************************************/
$newFile = new File;
```

12.1.2.3 Comment Entity

The Comment Entity Class and its properties are incredibly similar to the User and File Entity Classes. The comment entity portion will not have any descriptions for its property uses.

12.1.2.3.1 Indexes

Comment Index are used to find comments relatively quickly within the database.

12.1.2.3.1.1 Comment Id

commentId will store the id of the comment. commentId has get and set functions.

Listing 62: Comment Id Index

```
/**********************************************************
* Variable: commentId
**********************************************************
* Description: Stores the comments id
```

```
*****************************************************/
/**
* @OGM\Auto
*/
protected $commentId;


/*****************************************************
* Function: getCommentId()
*****************************************************
* Description: Returns the comment id.
*****************************************************/
function getCommentId(){
        return $this->commentId;
}


/*****************************************************
* Function: setCommentId($commentId)
*****************************************************
* Description: Sets the comment id
*****************************************************/
function setCommentId($commentId){
        $this->commentId = $commentId;
}
```

12.1.2.3.1.2 Comment Unique Id

commentUniqueId will store the auto-generated unique id for the comment. commentUniqueId has a get functions.

Listing 63: Comment Unique Id Index

```
/*****************************************************
* Variable: commentUniqueId
*****************************************************
* Description: Stores the comments unique id.
*****************************************************/
/**
* @OGM\Property
* @OGM\Index
*/
$commentUniqueId;


/*****************************************************
* Function: getCommentUniqueId()
```

```
*******************************************************
* Description: Returns the comments unique id.
*******************************************************/
function getCommentUniqueId(){
        return $this->commentUniqueId;
}
```

12.1.2.3.2 Properties

Comment Properties store information such as when the comment was made and what the comment text is.

12.1.2.3.2.1 Comment Type

commentType will store whether the comment is a file or user profile comment. This is user in co-ordination with commentTo to make sure the correct comments are being displayed. commentType has get and set functions.

Listing 64: Comment Type Property

```
/*******************************************************
* Variable: commentType
*******************************************************
* Description: Stores the type of comment
*******************************************************/
/**
* @OGM\Property
*/
protected $commentType;


/*******************************************************
* Function: getCommentType()
*******************************************************
* Description: Returns the type of comment this is
*******************************************************/
function getCommentType(){
        return $this->commentType;
}


/*******************************************************
* Function: setCommentType($commentType)
*******************************************************
* Description: Sets the comment type
*******************************************************/
function setCommentType($commentType){
        $this->commentType =$commentType;
```

```
}
```

12.1.2.3.2.2 Comment Date

commentDate will store a time stamp of when the comment was was made. commentDate has get and set functions.

Listing 65: Comment Date Property

```
/**********************************************************
* Variable: commentDate
**********************************************************
* Description: Stores the date the comment was made
**********************************************************/
/**
* @ODM\Property(format="date")
*/
protected $commentDate;


/**********************************************************
* Function: getCommentDate()
**********************************************************
* Description: Returns the date the comment was made
**********************************************************/
function getCommentDate(){
        return $this->commentDate;
}


/**********************************************************
* Function: setCommentDate($commentDate)
**********************************************************
* Description: Sets the date the comment was made
**********************************************************/
function setCommentDate($commentDate){
        $this->commentDate = $commentDate;
}
```

12.1.2.3.2.3 Comment Likes

commentLikes stores the total number of likes this comment has. commentLikes has get and set functions

Listing 66: Comment Likes Property

```
/**********************************************************
* Variable: commentLikes
```

```
/**********************************************************
* Description: Stores the total number of likes this comment
* has
**********************************************************/
/**
* @OGM\Property
*/
protected $commentLikes;


/**********************************************************
* Function: getCommentLikes()
**********************************************************
* Description: Returns the number of likes this comment has
**********************************************************/
function getCommentLikes(){
        return $this->commentLikes;
}


/**********************************************************
* Function: setCommentLikes
**********************************************************
* Description: Sets the total number of comment likes
**********************************************************/
function setCommentLikes($commentLikes){
        $this->commentLikes = $commentLikes;
}
```

12.1.2.3.2.4 Comment Dislikes

commentDislikes stores the total number of dislikes a comment has. commentDislikes has get and set functions.

Listing 67: Comment Dislikes Property

```
/**********************************************************
* Variable: commentDislikes
**********************************************************
* Description: Stores the total number of comment dislikes
**********************************************************/
/**
* @OGM\Property
*/
protected $commentDislikes;
```

```
/**********************************************************
* Function: getCommentDislikes()
**********************************************************
* Description: Returns the total number of dislikes this
* comment has
**********************************************************/
function getCommentDislikes(){
        return $this->commentDislikes;
}


/**********************************************************
* Function: setCommentDislikes($commentDislikes)
**********************************************************
* Description: Sets the total number of comment dislikes
**********************************************************/
function setCommentDislikes($commentDislikes){
        $this->commentDislikes = $commentDislikes;
}
```

12.1.2.3.2.5 Comment User

commentUser will store a user node. The user node stored is the user that made the comment. commentUser has get and set functions.

Listing 68: Comment User Property

```
/**********************************************************
* Variable: commentUser
**********************************************************
* Description: Stores the user that made the comment
**********************************************************/
/**
* @OGM\ManyToOne(relation="madeComment")
*/
protected $commentUser


/**********************************************************
* Function: getCommentUser()
**********************************************************
* Description: Returns the user that made the comment
**********************************************************/
function getCommentUser(){
        return $this->commentUser;
}
```

```
/*********************************************************
* Function: setCommentUser(User $commentUser)
*********************************************************
* Description: Sets the user that made the comment
*********************************************************/
function setCommentUser(User $commentUser){
        $this->commentUser = $commentUser;
}
```

12.1.2.3.2.6 Comment To

commentTo will get the id of the page that the comment belongs to. We use this will commentType to figure out what specific user or file page this comment belongs to.

Listing 69: Comment To Property

```
/*********************************************************
* Variable: commentTo
*********************************************************
* Description: Stores the id of the page the comment belongs
* to
*********************************************************/
/**
* @OGM\Property
*/
protected $commentTo;


/*********************************************************
* Function: getCommentTo()
*********************************************************
* Description: Returns the id of the page the comment was
* made on
*********************************************************/
function getCommentTo(){
        return $this->commentTo;
}


/*********************************************************
* Function: setCommentTo($commentTo)
*********************************************************
* Description: Sets the page id the comment was made on
*********************************************************/
function setCommentTo($commentTo){
        $this->commentTo = $commentTo;
```

```
}
```

12.1.2.3.2.7 Comment Text

commentText stores the actual text of the comment. commentText has get and set functions

Listing 70: Comment Text Property

```
/**********************************************************
* Variable: commentText
**********************************************************
* Description: Stores the text of the comment
**********************************************************/
/**
* @OGM\Property
*/
protected $commentText;


/**********************************************************
* Function: getCommentText()
**********************************************************
* Description: Returns the comments text
**********************************************************/
function getCommentText(){
        return $this->commentText;
}


/**********************************************************
* Function: setCommentText($commentText)
**********************************************************
* Description: Sets the comments text
**********************************************************/
function setCommentText($commentText){
        $this->commentText = $commentText;
}
```

12.1.2.3.2.8 Comment Edit Date

commentEditDate stores the date the comment was edited on. commentEditDate has get and set functions

Listing 71: Comment Edit Date Property

```
/**********************************************************
* Variable: commentEditDate
**********************************************************
```

```
* Description: Stores the date the commented was edited on
**********************************************************/
/**
* @OGM\Property(format="date")
*/
protected $commentEditDate;


/**********************************************************
* Function: getCommentEditDate()
**********************************************************
* Description: Returns the date the comment was edited on
**********************************************************/
function getCommentEditDate(){
        return $this->commentEditDate;
}


/**********************************************************
* Function: setCommentEditDate($commentEditDate)
**********************************************************
* Description: Sets the date the comment was edited on
**********************************************************/
function setCommentEditDate($commentEditDate){
        $this->commentEditDate = $commentEditDate;
}
```

12.1.2.3.2.9 Comment Reported

commentReported stores a boolean value to whether or not this comment needs adminstrative review. commentReported has get and set functions.

Listing 72: Comment Reported Property

```
/**********************************************************
* Variable: comment
**********************************************************
* Description: Stores
**********************************************************/
/**
* @OGM\Property
*/
protected $commentReported;


/**********************************************************
* Function: getCommentReported()
```

```
*******************************************************
* Description: Returns if the comment is reported
*******************************************************/
function getCommentReported(){
        return $this->commentReported;
}


/*******************************************************
* Function: setCommentReported($commentReported)
*******************************************************
* Description: Sets if the comment has been reported.
*******************************************************/
function setCommentReported($commentReported){
        $this->commentReported = $commentReported;
}
```

12.1.2.3.3 Arrays

The Comment Class does not contain any array properties. Only the File Class has array properties.

12.1.2.3.4 ArrayCollections

The Comment Class doesn't need to hold many relationships. The only ArrayCollections that it needs are for Users who liked and disliked the Comment.

12.1.2.3.4.1 Comment Likes By Users

commentLikesByUsers stores an ArrayCollection of the users that liked this comment. commentLikesByUsers has get, set, add, remove, and has functions

Listing 73: Comment Likes By Users Array Collection

```
/*******************************************************
* Variable: commentLikesByUsers
*******************************************************
* Description: Stores the users that liked this comment
*******************************************************/
/**
* @ODM\ManyToMany(relation="userLikedComment")
*/
protected $commentLikesByUsers;


/*******************************************************
* Function: getCommentLikesByUsers()
*******************************************************
* Description: Returns the users that liked this comment
*******************************************************/
```

```
function getCommentLikesByUsers(){
        return $this->commentLikesByUsers;
}


/*********************************************************
* Function: setCommentLikesByUsers(ArrayCollection $commentLikesByUsers
    )
*********************************************************
* Description: Sets the users that liked this comment
*********************************************************/
function setCommentLikesByUsers(ArrayCollection $commentLikesByUsers){
        $this->commentLikesByUsers = $commentLikesByUsers;
}

/*********************************************************
* Function: addCommentLikesByUsers(User $userLiked)
*********************************************************
* Description: Adds a new comment like by user
*********************************************************/
function addCommentLikesByUsers(User $userLiked){
        $this->commentLikesByUsers->add($userLiked);
}


/*********************************************************
* Function: removeCommentLikesByUsers(User $userLiked)
*********************************************************
* Description: Removes a user that liked the file
*********************************************************/
function removeCommentLikesByUsers(User $userLiked){
        $this->commentLikesByUsers->removeElement($userLikes);
}


/*********************************************************
* Function: hasCommentLikesByUsers(User $user)
*********************************************************
* Description: Checks if the user has liked the comment
*********************************************************/
function hasCommentLikesByUsers(User $user){
        return $this->commentLikesByUsers->contains($user);
}
```

12.1.2.3.4.2 Comment Dislikes By Users

commentDislikesByUsers will store an ArrayCollection of users who disliked the comment. commentDislikesByUsers has get, set, add, remove, and has functions.

Listing 74: Comment Dislikes By Users Array Collection

```
/**********************************************************
* Variable: commentDislikesByUsers
**********************************************************
* Description: Stores the users who disliked the comment
**********************************************************/
/**
* @OGM\ManyToMany(relation="userDislikedComment")
*/
protected $commentDislikesByUsers;


/**********************************************************
* Function: getCommentDislikesByUsers()
**********************************************************
* Description: Returns the users who disliked the comment
**********************************************************/
function getCommentDislikesByUsers(){
        return $this->commentDislikesByUsers;
}


/**********************************************************
* Function: setCommentDislikesByUsers(ArrayCollection
    $commentDislikesByUsers)
**********************************************************
* Description: Sets the users that disliked the comment
**********************************************************/
function setCommentDislikesByUsers(ArrayCollection
    $commentDislikesByUsers){
        $this->commentDislikesByUsers = $commentDislikesByUsers;
}


/**********************************************************
* Function: addCommentDislikesByUsers(User $userDisliked)
**********************************************************
* Description: Adds a new user that disliked the comment
**********************************************************/
function addCommentDislikesByUsers(User $userDisliked){
        $this->commentDislikesByUsers->add($userDisliked);
}
```

```
/*********************************************************
* Function: removeCommentDislikesByUsers(User $userDisliked)
*********************************************************
* Description: Removes a user that disliked the comment
*********************************************************/
function removeCommentDislikesByUsers(User $userDisliked){
        $this->commentDislikesByUsers->removeElement($userDisliked);
}


/*********************************************************
* Function: hasCommentDislikesByUsers(User $user)
*********************************************************
* Description: Checks if the user has disliked the comment
*********************************************************/
function hasCommentDislikesByUsers(User $user){
                return $this->commentDislikesByUsers->contains($user);
        }
```

12.1.2.3.5 Comment Entity Construction This section will breifly go over the construction of the Comment Entity.

12.1.2.3.5.1 Comment Construct

Listing 75: Comment Construct Function

```
function __construct(){
    /*********************************************************
    * Comment Unique Id will have its id automatically generated
    * here
    *********************************************************/

        $this->commentUniqueId = uniqid();

    /*********************************************************
    * Creates standard ArrayCollections waiting to be filled
    *********************************************************/

        $this->commentLikesByUsers = new ArrayCollection;
        $this->commentDislikesByUsers = new ArrayCollection;
}
/*********************************************************
* To Create a New Comment Entity
```

```
**********************************************************/
$newComment = new Comment;
```

12.1.2.4 Sample Entity

The sample entity class is here simply to understand how the class itself is created. To see the full construction of the User, File, and Comment Entity classes, they are available in the source code.

Listing 76: Sample Entity Class

```
<?php
//Gives us the correct namespace, as well as frameworks,
//needed to properly set up the class.
namespace HireVoice\Neo4j\divy-dev\Entity;
use Doctrine\Common\Collections\ArrayCollection;
use HireVoice\Neo4j\Annotation as OGM;

//The OGM entity will select the appropriate repository
//for the class

/**
* @OGM\Entity(repositoryClass="Repository\NewClassRepository")
*/
class NewClass{

//Protected Properties, Indexes, Arrays,
//and ArrayCollections go here.

    /**
    * @OGM\Auto
    */
    protected $newClassId;

    /**
    * @OGM\Property
    * @OGM\Index
    */
    protected $newClassUniqueId;

    /**
    * @OGM\Property
    * @OGM\Index
    */
    protected $newClassName;

    /**
```

```
    * @OGM\Property
    */
    protected $newClassInteractionCount;

    /**
    * @OGM\Property(format="date")
    */
    protected $newClassDate;

    /**
    * @OGM\Property(format="array")
    */
    protected $newClassArrayExample;

    /**
    * @OGM\ManyToOne(relationship="ownsNewClass")
    */
    protected $newClassOwner;

    /**
    * @OGM\ManyToMany(relationship="intereactedWithNewClass")
    */
    protected $newClassArrayCollectionExample;

    //The construct function is next.
    function __construct(){
        $this->newClassArrayExample = array();
        $this->newClassArrayCollectionExample =
        new ArrayCollection;

        $this->newClassUniqueId = uniqid();
    }

    //Get,Set,Add,Remove,and Has functions
    //as they are needed

    function getNewClassId(){
        return $this->newClassId;
    }

    function setNewClassId($newClassId){
        $this->newClassId = $newClassId;
    }

    function getNewClassUniqueId(){
        return $this->newClassUniqueId;
```

```
    }

    function getNewClassName(){
        return $this->newClassName;
    }

    function setNewClassName($newClassName){
        $this->newClassName = $newClassName;
    }

    function getNewClassInteractionCount(){
        return $this->newClassInteractionCount;
    }

    function setNewClassInteractionCount($newClassInteractionCount){
        $this->newClassInteractionCount = $newClassInteractionCount;
    }

    function getNewClassDate(){
        return $this->newClassDate;
    }

    function setNewClassDate($newClassDate){
        $this->newClassDate = $newClassDate;
    }

    function getNewClassArrayExample(){
        return $this->newClassArrayExample;
    }

    function addNewClassArrayExample($newArrayEntry){
        $this->newClassArrayExample[] = $newArrayEntry;
    }

    function setNewClassArrayExample(array $newClassArrayExample){
        $this->newClassArrayExample = array();
        foreach($newClassArrayExample as $newArrayEntry){
                $this->addNewClassArrayExample($newArrayEntry);
        }
    }

    function getNewClassArrayCollectionExample(){
        return $this->newClassArratCollectionExample;
    }

    function setNewClassArrayCollectionExample
```

```
    (ArrayCollection $newClassArrayCollectionExample){
        $this->newClassArratCollectionExampe
        = $newClassArrayCollectionExample;
    }

    function addNewClassArrayCollectionExample(Node $newNodeEntry){
        $this->newClassArratCollectionExampe->add($newNodeEntry);
    }

    function removeNewClassArrayCollectionExample(Node $removeNodeEntry
        ){
        $this->newClassArratCollectionExampe
        ->removeElement($removeNodeEntry);
    }

    function hasNewClassArrayCollectionExample(Node $hasThisNode){
        $this->newClassArratCollectionExampe->contains($hasThisNode);
    }
}
?>
```

12.1.3 Repositories

This section will go over the repositories used in the project. These repositories allow us to search our database with Cypher Queries. Each Entity has its own repository.

12.1.3.1 Divy Neo4JPHP-OGM Cypher Queries

In the formation of the backend of Divy's website, forming cypher queries in PHP proved to be rather difficult. However, by using the NEO4JPHP-OGM framework and extending its use, we can create incredibly useful queries in recommending files and users to Divy's members. The purpose of this section is to go over the NEO4JPHP-OGM Entity manager breifly and cover the cypher query commands used in Divy.

12.1.3.1.0.2 Entity Manager

The Entity Manager is what keeps track of the entities in the Neo4J database. We will see that by extending the base repository framework, we can use this entity manager in our own cypher queries.

Listing 77: Get Entity Manager Function

```
/**********************************************************
* Function: getEntityManager()
**********************************************************
* Description: Returns the entity manager that interacts
* and keeps track of entites within the Neo4J database.
**********************************************************/
```

```
getEntityManager()


/*********************************************************
* Example Description: Get the Entity Manager for the
* current repository.
*********************************************************/
$em = $this->getEntityManager();
```

12.1.3.1.0.3 Create Cypher Query

Listing 78: Create Cypher Query Function

```
/*********************************************************
* Function: createCypherQuery()
*********************************************************
* Description: Will create a new cypher query that can be
* used in searching our Neo4J database.
*********************************************************/
createCypherQuery()


/*********************************************************
* Example Description: Create a new cypher query after
* getting the entity manager
*********************************************************/
$em = $this->getEntityManager();
$startQuery = $em->createCypherQuery()
                 ->...
```

12.1.3.1.0.4 Start

Start will tell us what type of entities to be searching for on the site, whether it be Users, Files, or Comments.

Listing 79: Start Function

```
/*********************************************************
* Function: start('entity = node(:entity)')
*********************************************************
* Description: start will come after the createCypherQuery
* function, and determines which set of entites to search
* through
*********************************************************/
start('entity = node(:entity)')
```

```
/*********************************************************
* Example Description: Start the node search on all user
* nodes after creating the cypher query
*********************************************************/
$em = $this->getEntityManager();
$startQuery = $em->createCypherQuery()
                 ->start('user = node(:user)')
          ->...
```

12.1.3.1.0.5 Start With Node

Start with node will start the cypher query at a specific node based on the parameters given.

Listing 80: Start With Node Function

```
/*********************************************************
* Function: startWithNode('entity', $entityUser)
*********************************************************
* Description: will start the cypher query at the specific node
*********************************************************/
startWithNode('entity', $entityUser)


/*********************************************************
* Example Description: Start the cypher query search with
* the specific user given.
*********************************************************/
$em = $this->getEntityManager();
$startQuery = $em->createCypherQuery()
                 ->startWithNode('user', $userGiven)
                 ->...
```

12.1.3.1.0.6 Where

Where will only retrieve entities based if a specific entity property matches the clause.

Listing 81: Where Function

```
/*********************************************************
* Function: where('entity.entityProperty = :entityProperty')
*********************************************************
* Description: Where will only match entities that match the
* specific property give
*********************************************************/
where('entity.entityProperty = :entityProperty')
```

```
/*******************************************************
* Example Description: Get the users where there name is
* similar to the searched name
*******************************************************/
$em = $this->getEntityManager();
return $em->createCypherQuery()
        ->start('user = node(:user)')
        ->where('user.userName = :'.$userName.'')
            ->...
```

12.1.3.1.0.7 Match

Match will retrieve entities based on the relationships formed by the entites previously matched.

Listing 82: Match Function

```
/*******************************************************
* Function: match('(entity) -[:entityRelationShip]-> (otherEntity)')
*******************************************************
* Description: Match will only get the entities that have the
* entity relationship given.
*******************************************************/
match('(entity) -[:entityRelationShip]-> (otherEntity)')


/*******************************************************
* Example Description: Match the users that subscribed to
* the given users
*******************************************************/
$em = $this->getEntityManager();
return $em->createCypherQuery()
        ->startWithNode('user', $subscribedToUser)
        ->match('(user) -[:userSubscribers]-> (subscribers)')
            ->...
```

12.1.3.1.0.8 End

End will determine what entities that were matched should be retrieved. If there is a count clause within end, then we need an order command after.

Listing 83: End Function

```
/*******************************************************
* Function: end('entity')
*                       end('entity, count(*)')
*******************************************************
* Description: End will stop the cypher query will the
* results so far. If there is a count command we have to
```

```
* set the order.
*****************************************************/



/*****************************************************
* Example Description: End with all subscribers to the given
* user
*****************************************************/
$em = $this->getEntityManager();
return $em->createCypherQuery()
        ->startWithNode('user', $subscribedToUser)
        ->match('(user) -[:userSubscribers]-> (subscribers)')
         ->end('subscribers')
             ->...
```

12.1.3.1.0.9 Order

Determines in what order the entities should be returned if a list. This can be based on entities with the most of a specific relationship, or the total count of a property on an entity.

Listing 84: Order Function

```
/*****************************************************
* Function: order('count(*), entity.EntityProperty')
*****************************************************
* Description: Will order the result of the cypher query
* based on the entity property given. This can even give the
* entity with the most relationships based on the entities
* given
*****************************************************/
order('count(*), entity.EntityProperty')


/*****************************************************
* Example Description: Order the users returned based on the
* comment likes they have.
*****************************************************/
$em = $this->getEntityManager();
return $em->createCypherQuery()
        ->start('user = node(:user)')
        ->end('user, count(*)')
        ->order('count(*), user.userTotalCommentLikes')
          ->...
```

12.1.3.1.0.10 Get One

Will return the only match for the query result. Incredibly useful in searching based on entity

indexed parameters.

Listing 85: Get One Function

```
/**********************************************************
* Function: getOne()
**********************************************************
* Description: Will get a single entity that matched the
* cypher query
**********************************************************/
getOne()


/**********************************************************
* Example Description: get a single user based on the
* name given.
**********************************************************/
$em = $this->getEntityManager();
return $em->createCypherQuery()
        ->start('user = node(:user)')
        ->where('user.userName = :'.$userName.'')
        ->end('user')
        ->getOne();
```

12.1.3.1.0.11 Get List

Will return the list of matches for the cypher query.

Listing 86: Get List Function

```
/**********************************************************
* Function: getList()
**********************************************************
* Description: Will return a list of entities that matched
* the cypher query
**********************************************************/
getList()


/**********************************************************
* Example Description: Get a list of users that are most
* subscribed to.
**********************************************************/
$em = $this->getEntityManager();
return $em->createCypherQuery()
        ->start('user = node(:user)')
        ->end('user, count(*)')
        ->order('count(*), user.userTotalSubscribers')
```

```
        ->getList();
```

12.1.3.2 User Repository

The User Repository contains cypher queries used in finding users based on the parameters given.

12.1.3.2.1 Single User Entity Results

Single User Entity Results will be cypher query results that only return a single user as a result. This is useful in searching users by name, id, and email.

12.1.3.2.1.1 Find One User By Id

findOneUserById will find a single user based on the id given.

Listing 87: Find One User By Id Function

```
/**********************************************************
* Function: findOneUserById($userid)
**********************************************************
* Description: Creates a new cypher query that will start
* with all user nodes, and then match users with this id.
* Because there is only one id associated with each user,
* we can use getOne() to retrieve that single user.
**********************************************************/
public function findOneUserById($userId){
        return $em->createCypherQuery()
                ->start('user = node(:user)')
                ->where('user.userId = :'.$userId.'')
                ->end('user')
                ->getOne();
}


/**********************************************************
* Example Description: Returns the user based on the name
* given.
**********************************************************/
$userWithId123 = $userRepo->findOneUserById('123');
```

12.1.3.2.1.2 Find One User By Name

findOneUserByName will return a single user based on the name given

Listing 88: Find One User By Name Function

```
/**********************************************************
* Function: findOneUserByName($userName)
**********************************************************
```

```
* Description: This cypher query will return a single user
* based on the user name given
*****************************************************/
public function findOneUserByName($userName){
        return $em->createCypherQuery()
                ->start('user = node(:user)')
                ->where('user.userName = :'.$userName.'')
                ->end('user')
                ->getOne();
}


/*******************************************************
* Example Description: Returns a single user based on the
* name given.
*******************************************************/
$userNamedBill = $userRepo->findOneUserByName('Bob');
```

12.1.3.2.1.3 Find One User By Email

findOneUserByEmail will return a single user based on the email given.

Listing 89: Find One User By Email Function

```
/*******************************************************
* Function: findOneUserByEmail($userEmail)
*******************************************************
* Description: Returns a single user based on the email given
*******************************************************/

/*******************************************************
* Example Description: Given the specific email, return the
* user with that email.
*******************************************************/
$email = 'divydev@divy.com';
$userWithEmail = $userRepo->findOneUserByEmail($userEmail);
```

12.1.3.2.2 Multiple User Entity Results

Multiple User Entity Results will return a list of users based on the queries wanted.

12.1.3.2.2.1 Matching Strings For Relevancy

In order to match strings based on relevancy, the following utility function has been created.

Listing 90: Get String Regex User Function

```
public function getStringRegexUser($searchedName){
$pattern ='';
```

```
$searchedNames = explode(" ", $searchedName);
$i = 0;
$foreach($searchedName as $searchString){
  if(strlen($searchString)>=3){
    $substr = mb_substr($searchString,0,2);
    if($i = 0){
        $pattern = ':(?i)'.$substr.'*';
    }
    else{
                $pattern .= 'OR (user.userName =~ :(?i)'.$substr.'*)';
    }
  }
  else{
        if($i =0){
                $pattern = ':(?i)'.$searchString.'*';
        }
        else{
                $pattern .= 'OR (user.userName =~ :(?i)'.$searchString
                    .'*)';
        }
  }
  $i++;
}
return $pattern;
}
```

12.1.3.2.2.2 Find Most Relevant Users

findMostRelevantUsers will return a list of users in order of relatedness to the searched name.

Listing 91: Find Most Relevant Users Function

```
/*********************************************************
* Function: findMostRelevantUsers($searchedName)
*********************************************************
* Description: Returns the list of users in order of
* relevancy to the name searched
*********************************************************/
public function findMostRelevantUsers($searchedName){
  if($searchedName == ''){
    return $em->createCypherQuery()
        ->start('user = node(:user)')
        ->end('user');
        ->getList();
  }
  else{
    $searchPattern = getStringRegexUser($searchedName);
```

```
    return $em->createCypherQuery()
        ->start('user = node(:user)')
        ->where('user.userName =~ '.$searchPattern.'')
        ->end('user');
        ->getList();
  }
}

/*********************************************************
* Example Description: Finds the most relevant users to
* the name searched
*********************************************************/
$searchedName = 'Timmy';
$findUsersNamedLikedTimmy = $userRepo->findMostRelevantUsers(
    $searchedName);
```

12.1.3.2.2.3 Find Most Subscribed To Users

findMostSubscribedToUsers will return the users related to the searched name in order of the number of subscribers they have.

Listing 92: Find Most Subscribed To Function

```
/*********************************************************
* Function: findMostSubscribedTo($searchedName)
*********************************************************
* Description: Returns the most subscribed to users based
* on the searched name
*********************************************************/
public function findMostSubscribedToUsers($searchedName){
  if($searchedName == ''){
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->end('user, count(*)')
      ->order('count(*), user.userTotalSubscribers')
      ->getList();
  }
  else{
    $searchPattern = getStringRegexUser($searchedName)
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->where('user.userName =~ '.$searchPattern.'')
      ->end('user, count(*)')
      ->order('count(*), user.userTotalSubscribers')
      ->getList();
  }
}
```

```
/*******************************************************
* Example Description: Lists all users in order of the
* number of subscribers they have.
*******************************************************/
$searchedTerm = '';
$usersInOrderOfSubscribers = $userRepo->findMostSubscribedTo(
    $searchedTerm);
```

12.1.3.2.2.4 Find Users With Most File Views

findUsersWithMostFileViews will returns a list of users related to the searched name in order of the number of total file views they have.

Listing 93: Find Users With Most File Views Function

```
/*******************************************************
* Function: findUsersWithMostFileViews($searchedName)
*******************************************************
* Description: Returns a list of users related to the
* searched name in order of their total file views.
*******************************************************/
public function findUsersWithMostFileViews($searchedName){
  if($searchedName == ''){}
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->end('user, count(*)')
      ->order('count(*), user.userTotalOwnedFileViews')
      ->getList();
  else{
    $searchPattern = getStringRegexUser($searchedName);
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->where('user.userName =~ '.$searchPatten.'.*')
      ->end('user, count(*)')
      ->order('count(*), user.userTotalOwnedFileViews')
      ->getList();
  }
}

/*******************************************************
* Example Description: Returns the users related to the
* searched name in order of their file views
*******************************************************/
$searchedName = 'Kevin';
$usersNamedLikeKevinInOrderOfFileViews = $userRepo->
    findUsersWithMostFileViews($searchedName);
```

12.1.3.2.2.5 Find Users With Most File Likes

findUsersWithMostFileLikes will return a list of users related to the searched name in order of the number of total file views they have.

Listing 94: Find Users With Most File Likes Function

```
/*********************************************************
* Function: findUsersWithMostFileLikes($searchedName)
*********************************************************
* Description: Returns a list of related users in order
* of the number of total file likes they have
*********************************************************/
public function findUsersWithMostFileLikes($searchedName){
    if($searchName == ''){
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->end('user, count(*)')
      ->order('count(*), user.userTotalOwnedFileLikes')
      ->getList();
  }
  else{
    $searchPattern = getStringRegexUser($searchedName);
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->where('user.userName =~ '.$searchPatten.'')
      ->end('user, count(*)')
      ->order('count(*), user.userTotalOwnedFileLikes')
      ->getList();
  }
}
/*********************************************************
* Example Description: List all users in order of the
* users with the most file likes.
*********************************************************/
$searchedName = '';
$mostLikedUsers = $userRepo->findUsersWithMostFileLikes($seachedName);
```

12.1.3.2.2.6 Find Newest Users

findNewestUsers will return a list of related users in order of newest to oldest.

Listing 95: Find Newest Users Function

```
/*********************************************************
* Function: findNewestUsers($searchedName)
*********************************************************
* Description: Finds the Newest Users based on the searched
* name
```

```
*******************************************************/
public function findNewestUsers($searchedName){
  if($searchedName == '');{
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->end('user, count(*)')
      ->order('count(*), user.userRegisterDate')
      ->getList();
  }
  else{
    $searchPattern = getStringRegexUser($searchedName);
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->where('user.userName =~ '.$searchPattern.'')
      ->end('user, count(*)')
      ->order('count(*), user.userRegisterDate')
      ->getList();
        }
}
/*******************************************************
* Example Description: Find the newest users with the
* name searched
*******************************************************/
$searchedName = 'New Nelly';
$newestUsersNamedLikeNewNelly = $userRepo->findNewestUsers(
   $searchedName);
```

12.1.3.2.2.7 Find Users With Most Comment Likes

findUsersWithMostCommentLikes will return a list of related users in order of the number of comment likes they have.

Listing 96: Find Users With Most Comment Likes Function

```
/*******************************************************
* Function: findUsersWithMostCommentLikes($searchedName)
*******************************************************
* Description: Returns a list of related users based on
* the number of comment likes they have
*******************************************************/
public function findUsersWithMostCommentLikes($searchedName){
  if($searchedName == ''){
    return $em->createCypherQuery()
      ->start('user = node(:user)')
      ->end('user, count(*)')
      ->order('count(*), user.userTotalCommentLikes')
      ->getList();
```

```
  }
  else{
    $searchPattern = getStringRegexUser($searchedName);
      return $em->createCypherQuery()
        ->start('user = node(:user)')
        ->where('user.userName =~ '.$searchPattern.'')
        ->end('user, count(*)')
        ->order('count(*), user.userTotalCommentLikes')
        ->getList();
  }
}
/**********************************************************
* Example Description: Returns a list of all users in order
* of which users have the most comment likes
**********************************************************/
$searchedName = '';
$usersWithMostCommentLikes = $userRepo->findUsersWithMostCommentLikes(
    $searchedName);
```

12.1.3.2.2.8 Find Most Subscribed To Users After Subscribed To

findMostSubscribedToUsersAfterSubscribedTo will return a list of users that are most subscribed to by users who are subscribers of the subscribed to user.

Listing 97: Find Most Subscribed To Users After Subscribed To

```
/**********************************************************
* Function: findMostSubscribedToUsersAfterSubribedTo(User
    $subscribedToUser)
**********************************************************
* Description: Returns a list of the most subscribed to user
* by subscribers of this user. It will skip the first result
* as the most subscribed to user will be the subscribedToUser
**********************************************************/
public function findMostSubscribedToUsersAfterSubribedTo(User
    $subscribedToUser){
        return $em->createCypherQuery()
                ->startWithNode('user', $subscribedToUser)
                ->match('(user) -[:userSubscribers]-> (subscribers)')
                ->match('(subscribers) -[:userSubscribedTo]-> (
                    subscribedToUser)')
                ->end('subscribedToUsers, count(*)')
                ->order('count(*), subscribers.userSubscribedTo')
                ->skip(1)
                ->getList();
}
```

```
/*******************************************************
* Example Description: Returns the most recommended
* users to subscribe to based on the user subsribed to
*******************************************************/
$userSubscribedTo = $thisUser;
$mostSubscribedToAfter = $userRepo->
    findMostSubscribedToUsersAfterSubscribedTo($subscribedToUser);
```

12.1.3.2.2.9 Find Most Subscribed To Users With File Like

Listing 98: Find Most Subscribed To Users With File Like Function

```
/*******************************************************
* Function: findMostSubscribedToUsersWithFileLike(File $file)
*******************************************************
* Description: Returns the most subscribed to user by users
* who like this file
*******************************************************/
public function findMostSubscribedToUsersWithFileLike(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserLikes] -> (usersWhoLiked)')
                ->match('(usersWhoLiked) -[:userSubscribedTo] -> (
                    subscribedToUsers)')
                ->end('subscribedToUsers, count(*)')
                ->order('count(*), usersWhoLiked.userSubscribedTo')
                ->getList();
}

/*******************************************************
* Example Description: Returns the most subscribed to users
* after a file like
*******************************************************/
$fileLiked = $thisFile;
$mostSubscribedToAfterFileLike = $userRepo->
    findMostSubscribedToUsersWithFileLike($fileLiked);
```

12.1.3.2.2.10 Find Most Subscribed To Users With File View

findMostSubscribedToUsersWithFileView will return a list of users that are most subscribed to by users who viewed this file.

Listing 99: Find Most Subscribed To Users With File View Function

```
/*******************************************************
```

```
* Function: findMostSubscribedToUsersWithFileView(File $file)
*********************************************************
* Description: Returns the most subscribed to users by
* users who viewed this file
*********************************************************/
public function findMostSubscribedToUsersWithFileView(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserViews] -> (usersWhoViewed)')
                ->match('(usersWhoViewed) -[:userSubscribedTo] -> (
                    subscribedToUsers)')
                ->end('subscribedToUsers, count(*)')
                ->order('count(*), usersWhoViewed.userSubscribedTo')
                ->getList();
}

/*********************************************************
* Example Description: Return the most subscribed to users
* after a file view.
*********************************************************/
$fileViewed = $thisFile;
$usersSubscribedToAfterFileView = $userRepo->
   findMostSubscribedToUsersWithFileView(File $file);
```

12.1.3.2.2.11 Find Most Subscribed To Users With File Dislike

findMostSubscribedToUsersWithFileDislike will return the most subscribed to users by users who disliked the file.

Listing 100: Find Most Subscribed To Users With File Dislike

```
/*********************************************************
* Function: findMostSubscribedToUsersWithFileDislike(File $file)
*********************************************************
* Description: Returns a list of the users with the most
* subscribers who disliked this file
*********************************************************/
public function findMostSubscribedToUsersWithFileDislike(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserDislikes] -> (
                    usersWhoDisliked)')
                ->match('(usersWhoDisliked) -[:userSubscribedTo] -> (
                    subscribedToUsers)')
                ->end('subscribedToUsers, count(*)')
                ->order('count(*), usersWhoDisliked.userSubscribedTo')
                ->getList();
```

```
}

/*******************************************************
* Example Description: Get the list of the most subscribed
* to users after a file dislike.
*******************************************************/
$fileDisliked = $thisFile;
$usersSubscribedToAfterFileDislike = $userRepo->
    findMostSubscribedToWithFileDislike($fileDisliked);
```

12.1.3.3 File Repository

The File Repository contains cypher queries used in finding files based on the paramenters given.

12.1.3.3.1 Single File Entity Results

Single File Entity Results will be cypher query results that return a single file. This is useful in finding the file by its id, and checking if there is a file with that name and type already.

12.1.3.3.1.1 Find One File By Name And Type

findOneFileByNameAndType exists because we want users to be able to have the same name for different types of files. However, if one type of file has many files with the same name searches become more difficult.

Listing 101: Find One File By Name And Type Function

```
/*******************************************************
* Function: findOneFileByNameAndType($fileName, $fileType)
*******************************************************
* Description: Returns a single file by the name given
*******************************************************/
public function findOneByFileNameAndType($fileName, $fileType){
        return $em->createCypherQuery()
                ->start('file = node(:file)')
                ->where('file.fileName = :'.$fileName.'')
                ->where('file.fileType = :'.$fileType.'')
                ->end('file')
                ->getOne();
}
/*******************************************************
* Example Description: Find the file by the name given.
*******************************************************/
$fileType = 'jpg'
$fileName = 'RadRandy';
$radRandyJpgFile = $fileRepo->findOneFileByNameAndType($fileName,
    $fileType);
```

12.1.3.3.1.2 FindOneFileById

findOneFileById will get a single file by the given id.

Listing 102: Find One File By Id Function

```
/**********************************************************
* Function: findOneFileById($fileId)
**********************************************************
* Description: Returns a single file based on the id given
**********************************************************/
public function findOneFileById($fileId){
        return $em->createCypherQuery()
                ->start('file = node(:file)')
                ->where('file.fileId = :'.$fileId.'')
                ->end('file')
                ->getOne();
}


/**********************************************************
* Example Description: Returns the file based on the id
* given
**********************************************************/
$fileId = $givenFileId;
$fileWithThisId = $fileRepo->findOneFileById($fileId);
```

12.1.3.3.2 Multiple File Entity Results

Multiple File Entity Results will return a list of relevant information on the file searched or the file given.

12.1.3.3.2.1 Matching Strings For Relevancy

Similar to the user repository, there is a utility function to generate the where clause for matching relevant users.

Listing 103: Get String Regex File Function

```
public function getStringRegexFile($searchedName){
$pattern ='';
$searchedNames = explode(" ", $searchedName);
$i = 0;
$foreach($searchedName as $searchString){
  if(strlen($searchString)>=3){
    $substr = mb_substr($searchString,0,2);
    if($i = 0){
        $pattern = ':(?i)'.$substr.'*';
    }
    else{
```

```
                $pattern .= 'OR (file.fileName =~ :(?i).$substr.'*)';
    }
  }
  else{
    if($i =0){
        $pattern = ':(?i).$searchString.'*';
    }
    else{
        $pattern .= 'OR (file.userfile =~ :(?i).$searchString.'*)';
    }
  }
  $i++;
}
return $pattern;
}
```

12.1.3.3.2.2 Find Most Relevant Files

findMostRelevantFiles will return the most relevant files based on the name given, whether the file is free, and the files categories and type.

Listing 104: Find Most Relevant Files Function

```
/*********************************************************
* Function: findMostRelevantFiles($searchedTerm, $fileIsPaid,
   $fileCategory, $fileType)
*********************************************************
* Description: Returns the most relevant files to the searched
* name and other parameters
*********************************************************/
public function findMostRelevantFiles($searchedTerm, $fileIsPaid,
   $fileCategory, $fileType){
  if($fileIsPaid == ''){
        $fileIsPaid = *;
  }
  if($fileCategory == ''){
        $fileCategoryClause = 'file.fileCategory = :*';
  }
  else{
        $fileCategoryClause = 'ANY (x In file.fileCategory WHERE x ="'.
           $fileCategory.'")';
  }
  if($fileType == ''){
        $filetype = *;
  }
  if($searchedTerm == ''){
    return $em->createCypherQuery()
```

```
      ->start('file = node(:file)')
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
      ->where($fileCategoryClause)
      ->where('file.fileType = :'.$fileType.'')
      ->end('file, count(*)')
      ->getList();
  }
  else{
    $searchPattern = getStringRegexFile($searchedName);
    return $em->createCypherQuery()
      ->start('file = node(:file)')
      ->where('file.fileName =~ :'.$searchPattern)
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
      ->where($fileCategoryClause)
      ->where('file.fileType = :'.$fileType.'')
      ->end('file, count(*)')
      ->getList();
  }
}


/**********************************************************
* Example Description: Find the most relevant free files to
* the searched term
**********************************************************/
$searchedTerm = 'PHP Programming';
$fileIsPaid = 'Free';
$fileCategory = '';
$fileType = '';
$freeFilesRelatedToPHPProgramming = $fileRepo->findMostRelevantFiles(
   $searchedTerm, $fileIsPaid, $fileCategory, $fileType);
```

12.1.3.3.2.3 Find Most Liked Files

findMostLikedFiles will return a list of the most liked files based on the parameters given.

Listing 105: Find Most Liked Files Function

```
/**********************************************************
* Function: findMostLikedFiles($searchedTerm, $fileIsPaid,
   $fileCategory, $fileType)
**********************************************************
* Description: Finds the most liked files based on the
* parameters passed.
**********************************************************/
public function findMostLikedFiles($searchedTerm, $fileIsPaid,
   $fileCategory, $fileType){
```

```
  if($fileIsPaid == ''){
        $fileIsPaid = *;
  }
        if($fileCategory == ''){
  $fileCategoryClause = 'file.fileCategory = :*';
  }
  else{
        $fileCategoryClause = 'ANY (x In file.fileCategory WHERE x
           ="'.$fileCategory.'") ';
  }
  if($fileType == ''){
        $filetype = *;
  }
  if($searchedTerm == ''{
    return $em->createCypherQuery()
      ->start('file = node(:file)')
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
      ->where($fileCategoryClause )
      ->where('file.fileType = :'.$fileType.'')
      ->end('file, count(*)')
      ->order('count(*), file.fileDate')
      ->getList();
  }
  else{
    $searchPattern = getStringRegexFile($searchedName);
    return $em->createCypherQuery()
      ->start('file = node(:file)')
      ->where('file.fileName =~ :'.$searchPattern)
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
      ->where($fileCategoryClause )
      ->where('file.fileType = :'.$fileType.'')
      ->end('file, count(*)')
      ->order('count(*), file.fileLikes')
      ->getList();
  }
}

/*******************************************************
* Example Description: Find the most liked paid files that
* are in mp4 format.
*******************************************************/
$searchedTerm = '';
$fileIsPaid = 'Paid';
$fileCategory = '';
$fileType = 'mp4';
```

```
$mostLikedPaidMp4Files = $fileRepo->findMostLikedFiles($searchedTerm,
    $fileIsPaid, $fileCategory, $fileType);
```

12.1.3.3.2.4 Find Most Viewed Files

findMostViewedFiles will return a list of the most viewed files based on the parameters given

Listing 106: Find Most Viewed Files Function

```
/*********************************************************
* Function: findMostViewedFiles($searchedTerm, $fileIsPaid,
    $fileCategory, $fileType
*********************************************************
* Description: Returns the most viewed files based on
* the parameters given
*********************************************************/
public function findMostViewedFiles($searchedTerm, $fileIsPaid,
    $fileCategory, $fileType){
  if($fileIsPaid == ''){
        $fileIsPaid = *;
  }
  if($fileCategory == ''){
       $fileCategoryClause = 'file.fileCategory = :*';
  }
  else{
       $fileCategoryClause = 'ANY (x In file.fileCategory WHERE x ="'.
          $fileCategory.'") ';
  }
  if($fileType == ''){
        $filetype = *;
  }
  if($searchedTerm == ''{
    return $em->createCypherQuery()
      ->start('file = node(:file)')
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
      ->where($fileCategoryClause )
      ->where('file.fileType = :'.$fileType.'')
      ->end('file, count(*)')
      ->order('count(*), file.fileDate')
      ->getList();
  }
  else{
    $searchPattern = getStringRegexFile($searchedName);
    return $em->createCypherQuery()
      ->start('file = node(:file)')
      ->where('file.fileName =~ :'.$searchPattern)
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
```

```
        ->where($fileCategoryClause )
        ->where('file.fileType = :'.$fileType.'')
        ->end('file, count(*)')
        ->order('count(*), file.fileViews')
        ->getList();
    }
}
/*********************************************************
* Example Description: Returns the most viewed comedy files.
*********************************************************/
$searchedTerm = '';
$fileIsPaid = '';
$fileCategory = 'Comedy';
$fileType = '';
$mostViewedComedyFiles = $fileRepo->findMostViewedFiles($searchedTerm,
    $fileIsPaid, $fileCategory, $fileType);
```

12.1.3.3.2.5 Find Newest Files

findNewestFiles will return a list of the newest files based on the relatedness to the parameters given.

Listing 107: Find Newest Files Function

```
/*********************************************************
* Function: findNewestFiles($searchedTerm, $fileIsPaid, $fileCategory,
    $fileType)
*********************************************************
* Description: Returns the newest files based on the
* parameters passed.
*********************************************************/
public function findNewestFiles($searchedTerm, $fileIsPaid,
    $fileCategory, $fileType){
  if($fileIsPaid == ''){
        $fileIsPaid = *;
  }
  if($fileCategory == ''){
        $fileCategoryClause = 'file.fileCategory = :*';
  }
  else{
        $fileCategoryClause = 'ANY (x In file.fileCategory WHERE x ="'.
            $fileCategory.'") ';
  }
  if($fileType == ''){
        $filetype = *;
  }
  if($searchedTerm == ''{
```

```
    return $em->createCypherQuery()
      ->start('file = node(:file)')
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
      ->where($fileCategoryClause )
      ->where('file.fileType = :'.$fileType.'')
      ->end('file, count(*)')
      ->order('count(*), file.fileDate')
      ->getList();
  }
  else{
    $searchPattern = getStringRegexFile($searchedName);
    return $em->createCypherQuery()
      ->start('file = node(:file)')
      ->where('file.fileName =~ :'.$searchPattern)
      ->where('file.fileIsPaid = :'.$fileIsPaid.'')
      ->where($fileCategoryClause )
      ->where('file.fileType = :'.$fileType.'')
      ->end('file, count(*)')
      ->order('count(*), file.fileDate')
      ->getList();
  }
}

/*********************************************************
* Example Description: Find the newest files based on the
* the searched name only.
*********************************************************/
$searchedTerm = 'AJAX Tutorial';
$fileIsPaid = '';
$fileCategory = '';
$fileType = '';
$findNewestAJAXTutorialFiles = $fileRepo->findNewestFiles($searchedTerm
    , $fileIsPaid, $fileCategory, $fileType)
```

12.1.3.3.2.6 Find Most Viewed Files After File View

Listing 108: Find Most Viewed Files After File View Function

```
/*********************************************************
* Function: findMostViewedFilesAfterFileView(File $file)
*********************************************************
* Description:Returns the most viewed files by users who
* also viewed this file.
*********************************************************/
public function findMostViewedFilesAfterFileView(File $file){
```

```
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserViews] -> (usersWhoViewed)')
                ->match('(usersWhoViewed) -[:userFileViews] -> (
                    viewedFiles)')
                ->end('viewedFiles, count(*)')
                ->order('count(*), usersWhoViewed.userFileViews')
                ->skip(1)
                ->getList();

}

/**********************************************************
* Example Description: Get the most viewed files by users
* who also viewed this file.
**********************************************************/
$fileViewed = $thisFile;
$filesMostViewedAfterFileView = $fileRepo->
   findMostViewedFilesAfterFileView($fileViewed);
```

12.1.3.3.2.7 Find Most Viewed Files After File Like

findMostViewedFilesAfterFileLike will return the most viewed files by users who liked the file.

Listing 109: Find Most Viewed Files After File Like Function

```
/**********************************************************
* Function: findMostViewedFilesAfterFileLike(File $file)
**********************************************************
* Description: Finds the most viewed files after a user
* likes a file.
**********************************************************/
public function findMostViewedFilesAfterFileLike(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserLikes] -> (usersWhoLiked)')
                ->match('(usersWholiked) -[:userFileViews] -> (
                    viewedFiles)')
                ->end('viewedFiles, count(*)')
                ->order('count(*), usersWhoLiked.userFileViews')
                ->getList();
}

/**********************************************************
* Example Description: Find the most viewed files by users
* who liked this file.
**********************************************************/
```

```
$likedFile = $thisFile;
$mostViewedAfterFileLike = findMostViewedFilesAfterFileLike($likedFile)
    ;
```

12.1.3.3.2.8 Find Most Viewed Files After File Dislike

findMostViewedFilesAfterFileDislike will return the most viewed files by users who disliked the file.

Listing 110: Find Most Viewed Files After File Dislike Function

```
/**********************************************************
* Function: findMostViewedFilesAfterFileDislike(File $file)
**********************************************************
* Description: Returns the most viewed files by users who
* disliked the file.
**********************************************************/
public function findMostViewedFilesAfterFileDislike(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserDislikes] -> (
                    usersWhoDisliked)')
                ->match('(usersWhoDisliked) -[:userFileViews] -> (
                    viewedFiles)')
                ->end('viewedFiles, count(*)')
                ->order('count(*), usersWhoDisliked.userFileViews')
                ->getList();
}

/**********************************************************
* Example Description: Get the most viewed files after
* a file dislike
**********************************************************/
$fileDisliked = $thisFile;
$mostViewedFilesAfterDislike = $fileRepo->
    findMostViewedFilesAfterFileDislike($fileDisliked);
```

12.1.3.3.2.9 Find Most Liked Files After File View

findMostLikedFilesAfterFileView will return the most liked files by users who viewed the file.

Listing 111: Find Most Liked Files After File View Function

```
/**********************************************************
* Function: findMostLikedFilesAfterFileView(File $file)
**********************************************************
* Description: Returns the most viewed files by users
* who disliked the file.
**********************************************************/
```

```
public function findMostLikedFilesAfterFileView(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserViews] -> (usersWhoViewed)')
                ->match('(usersWhoViewed) -[:userFileLikes] -> (
                    likedFiles)')
                ->end('likedFiles, count(*)')
                ->order('count(*), usersWhoViewed.userFileLikes')
                ->getList();
}

/**********************************************************
* Example Description: Get the most liked files by users
* who viewed this file.
**********************************************************/
$viewedFile = $thisFile;
$mostLikedFilesAfterFileView = $fileRepo->
   findMostLikedFilesAfterFileView($viewedFile);
```

12.1.3.3.2.10 Find Most Liked Files After File Like

findMostLikedFilesAfterFileLike will return the most liked files by users who liked the file passed.

Listing 112: Find Most Liked Files After File Like Function

```
/**********************************************************
* Function: findMostLikedFilesAfterFileLike(File $file)
**********************************************************
* Description: Returns the most liked files by users who
* liked this file.
**********************************************************/
public function findMostLikedFilesAfterFileLike(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserLikes] -> (usersWhoLiked)')
                ->match('(usersWhoLiked) -[:userFileLikes] -> (
                    likedFiles)')
                ->end('likedFiles, count(*)')
                ->order('count(*), usersWhoLiked.userFileLikes')
                ->skip(1)
                ->getList();
}


/**********************************************************
* Example Description: Get the most liked files by users
* who liked this file.
```

```
*******************************************************/
$likedFile = $thisFile;
$mostLikedFilesByUsersWhoLikedFile = $fileRepo->
    findMostLikedFilesAfterFileLike($likedFile);
```

12.1.3.3.2.11 Find Most Liked Files After File Dislike

findMostLikedFilesAfterFileDislike will return the most liked files by users who disliked the file passed.

Listing 113: Find Most Liked Files After File Dislike Function

```
/*******************************************************
* Function: findMostLikedFilesAfterFileDislike(File $file)
*******************************************************
* Description: Returns the most liked files by users who
* disliked this file.
*******************************************************/
public function findMostLikedFilesAfterFileDislike(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileUserDislikes] -> (
                    usersWhoDisliked)')
                ->match('(usersWhoDisliked) -[:userFileLikes] -> (
                    likedFiles)')
                ->end('likedFiles, count(*)')
                ->order('count(*), usersWhoDisliked.userFileLikes')
                ->getList();
}

/*******************************************************
* Example Description: Find the most liked files after a
* a user dislikes the file.
*******************************************************/
$fileDisliked = $thisFile;
$mostLikedAfterFileDislike = $fileRepo->
    findMostLikedFilesAfterFileDislike($fileDisliked);
```

12.1.3.3.2.12 Find Most Liked Files After Subscribed To

findMostLikedFilesAfterSubscribedTo will return the most liked files by users who subscribed to this user.

Listing 114: Find Most Liked Files After Subscribed To Function

```
/*******************************************************
* Function: findMostLikedFilesAfterSubscribedTo(User $user)
*******************************************************
```

```
* Description: Returns the most liked files by subscribers.
*****************************************************/
public function findMostLikedFilesAfterSubscribedTo(User $user){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->startWithNode('user', $user)
                ->match('(user) -[:userSubscribers] -> (
                    usersWhoSubscribed)')
                ->match('(usersWhoSubscribed) -[:userFileLikes] -> (
                    file)')
                ->end('file, count(*)')
                ->order('count(*), usersWhoSubscribed.userFileLikes')
                ->getList();
}

/*****************************************************
* Example Description: Get the most liked files by
* this users subscribers
*****************************************************/
$userSubscribedTo = $thisUser;
$mostLikedFilesAfterSubscribedTo =$fileRepo->
    findMostLikedFilesAfterSubscribedTo($userSubscribedTo);
```

12.1.3.3.2.13 Find Most Viewed Files After Subscribed To

findMostViewedFilesAfterSubscribedTo will return the most viewed files by users who subscribed to this user.

Listing 115: Find Most Viewed Files After Subscribed To Function

```
/*****************************************************
* Function: findMostViewedFilesAfterSubscribedTo(User $user)
*****************************************************
* Description: Returns the most viewed files by this
* users subscribers
*****************************************************/
public function findMostViewedFilesAfterSubscribedTo(User $user){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->startWithNode('user', $user)
                ->match('(user) -[:userSubscribers] -> (
                    usersWhoSubscribed)')
                ->match('(usersWhoSubscribed) -[:userFileViews] -> (
                    file)')
                ->end('file, count(*)')
                ->order('count(*), usersWhoSubscribed.userFileViews')
                ->getList();
```

```
}


/*******************************************************
* Example Description: Get the most viewed files from this
* users subscribers
*******************************************************/
$userSubscribedTo = $thisUser;
$mostViewedAfterSubscribedTo = $fileRepo->
    findMostViewedFilesAfterSubscribedTo($userSubscribedTo);
```

12.1.3.3.2.14 Find Most Liked User Files

findMostLikedUserFiles will return a list of the most liked files a user has.

Listing 116: Find Most Liked User Files Function

```
/*******************************************************
* Function: findMostLikedUserFiles(User $user)
*******************************************************
* Description: Returns the most liked user files.
*******************************************************/
public function findMostLikedUserFiles(User $user){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->startWithNode('user', $user)
                ->match('(user) -[:userOwnedFiles] -> (file)')
                ->end('file, count(*)')
                ->order('count(*), file.fileLikes')
                ->getList();
}

/*******************************************************
* Example Description: Get this users most liked files.
*******************************************************/
$userWhoOwnsFiles = $thisUser;
$mostLikedUserFiles = $fileRepo->findMostLikedUserFiles(
    $userWhoOwnsFile);
```

12.1.3.3.2.15 Find Most Viewed User Files

findMostViewedUserFiles will return a list of the most viewed files a user has.

Listing 117: Find Most Viewed User Files Function

```
/*******************************************************
* Function: findMostViewedUserFiles(User $user)
*******************************************************
```

```
* Description: Returns the most viewed files this user owns
*****************************************************/
public function findMostViewedUserFiles(User $user){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->startWithNode('user', $user)
                ->match('(file) -[:userOwnedFiles] -> (file)')
                ->end('file, count(*)')
                ->order('count(*), file.fileViews')
                ->getList();
}

/*******************************************************
* Example Description: Get the most viewed files a user has
*******************************************************/
$userWhoOwnsFiles = $thisUser;
$mostViewedUserFiles = $fileRepo->findMostViewedUserFiles(
    $userWhoOwnsFiles);
```

12.1.3.3.2.16 Find Newest User Files

findNewestUserFiles will return a list of the most viewed files a user has.

Listing 118: Find Newest User Files Function

```
/*******************************************************
* Function: findNewestUserFiles(User $user)
*******************************************************
* Description: Returns the newest files a user owns.
*****************************************************/
public function findNewestUserFiles(User $user){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->startWithNode('user', $user)
                ->match('(file) -[:userOwnedFiles] -> (file)')
                ->end('file, count(*)')
                ->order('count(*), file.fileUploadDate')
                ->getList();
}


/*******************************************************
* Example Description: Get the newest files this user owns
*******************************************************/
$userWhoOwns = $thisUser;
$newestUserFiles = $fileRepo->findNewestUserFiles($userWhoOwns);
```

12.1.3.4 Comment Repository

The Comment Repository contains a single cypher query used in finding the comment by its id.

12.1.3.4.1 Single Comment Entity Results

Single Entity Comment Results will return a single comment based on the parameters

12.1.3.4.1.1 Find One Comment By Id

findOneCommentById will return a single comment entity

Listing 119: Find One Comment By Id Function

```
/**********************************************************
* Function: findOneCommentById($commentId)
**********************************************************
* Description: Returns a single comment based on this
* id
**********************************************************/
public function findOneCommentById($commentId){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->start('comment = node(:comment)')
                ->where('comment.commentId = :'.$commentId.'')
                ->end('comment')
                ->getOne();
}
```

12.1.3.4.2 Multiple Comment Entity Results

The Multiple Comment Entity Results will return the comments based on the file or user passed as a parameter.

12.1.3.4.2.1 Find Newest User Profile Comments

findNewestUserProfileComments will return the newest comments on a users profile

Listing 120: Find Newest User Profile Comments Function

```
/**********************************************************
* Function: findNewestUserProfileComments(User $user)
**********************************************************
* Description: Returns the newest profile comments
**********************************************************/
public function findNewestUserProfileComments(User $user){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->startWithNode('user', $user)
                ->match('(user) -[:userProfileComments] -> (
                    profileComments)')
```

```
                ->end('profileComments, count(*)')
                ->order('count(*), profileComments.commentDate')
                ->getList();
}

/**********************************************************
* Example Description: Get this users profile page comments
* from newest to oldest
**********************************************************/
$userOwnsProfile = $thisUser;
$newestProfileComments = $commentRepo->findNewestUserProfileComments(
    $userOwnsProfile);
```

12.1.3.4.2.2 Find Most Liked User Profile Comments

findMostLikedUserProfileComments will return the most liked comments on a users profile.

Listing 121: Find Most Liked User Profile Comments Function

```
/**********************************************************
* Function: findMostLikedUserProfileComments(User $user)
**********************************************************
* Description: Returns the most liked comments on this
* users profile page.
**********************************************************/
public function findMostLikedUserProfileComments(User $user){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                                ->startWithNode('user', $user)
                                ->match('(user) -[:userProfileComments]
                                    -> (profileComments)')
                                ->end('profileComments, count(*)')
                                ->order('count(*), profileComments.
                                    commentLikes')
                                ->getList();
}

/**********************************************************
* Example Description: Get this users profile most liked
* comments
**********************************************************/
$userOwnsProfile = $thisUser;
$mostLikedProfileComments = $commentRepo->
    findMostLikedUserProfileComments($userOwnsProfile);
```

12.1.3.4.2.3 Find Newest File Comments

findNewestFileComments will return the newest comments on a file

Listing 122: Find Newest File Comments Function

```
/**********************************************************
* Function: findNewestFileComments(File $file)
**********************************************************
* Description: Returns the newest comments users have made
* on a file
**********************************************************/
public function findNewestFileComments(File $file){
        $em = $this->getEntityManager();
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileComments] -> (fileComments)')
                ->end('fileComments, count(*)')
                ->order('count(*), fileComments.commentDate')
                ->getList();
}

/**********************************************************
* Example Description: Get this files newest comments
**********************************************************/
$fileWithComments = $thisFile;
$newestFileComments = $commentRepo->findNewestFileComments(
    $fileWithComments);
```

12.1.3.4.2.4 Find Most Liked File Comments

findMostLikedFileComments will return the most liked comments on a file

Listing 123: Find Most Liked File Comments Function

```
/**********************************************************
* Function: findMostLikedFileComments(File $file)
**********************************************************
* Description: Returns the most liked comments users have
* made on this file.
**********************************************************/
public function findMostLikedFileComments(File $file){
        return $em->createCypherQuery()
                ->startWithNode('file', $file)
                ->match('(file) -[:fileComments] -> (fileComments)')
                ->end('fileComments, count(*)')
                ->order('count(*), fileComments.commentLikes')
                ->getList();
}


/**********************************************************
```

```
* Example Description: Get the most liked comments a file
* has.
**********************************************************/
$fileWithComments = $thisFile;
$mostLikedFileComments = $commentRepo->findMostLikedFileComments(
    $fileWithComments)
```

12.1.3.5 Sample Repository

The purpose of this section is to show the full set-up of these repositories. The following repository example can be used in coordination with the Sampe Entity discussed in the Sample Entity section.

12.1.3.5.1

Listing 124: Sample Repository Class

```
<?php
/**********************************************************
* Gives us the correct namespace, as well as extending
* the NEO4JPHP-OGM BaseRepository Class.
**********************************************************/
namespace Repository;
use HireVoice\Neo4j\Repository as BaseRepository;

class NewClassRepository extends BaseRepository
{
    /**********************************************************
    * Find a single newClass node by the Id passed.
    **********************************************************/
    public function findOneNewClassById($newClassId){
        return $em->createCypherQuery()
                    ->start('newclass = node(:newclass)')
                ->where('newclass.newClassId =:'.$newClassId.'');
                ->end('newclass')
                ->getOne();
    }



  /**********************************************************
  * Return the most related newClass entities by the number of
  * times the have been interacted with.
  **********************************************************/
    public function findOtherNewClassNodesByInteractionCountAndName(
        $newClassName){
```

```
        return $em->createCypherQuery()
                ->start('newclass = node(:newclass)')
            ->where('newclass.newClassName =~ :(?).*'.$newClassName
                .'.*')
            ->end('newclass, count(*)')
            ->order('count(*), newclass.newClassInteractionCount)
            ->getList();
    }
}
?>

/*******************************************************
* Example of Creating a New Repository Query
*******************************************************/
$em = $this->get('hirevoice.neo4j.entity_manager');
$newClassRepo = $em->getRepository('Entity\\NewClass');
$newClassId = '123';
$newClassName = 'newClassName';

$newClassNodeById = $newClassRepo->findOneNewClassById($newClassId);
$newClassNodesByNameAndInteractionCount = $newClassRepo->
    findOtherNewClassNodesByInteractionCountAndName($newClassName);
```

12.1.4 Testing

This section will cover the testing of the various entities and repositories used. Test cases will be held in an Test Case that extends the NEO4J-PHP-OGM framework.

12.1.4.1 Creating New Test Cases

The following source code shows the set up of test cases for divy. This portion will create static entities and store values for them. There is no persistance within the entity manager in what happens in these test cases, so the Neo4J database wont be saved after the test cases. Though the relationship between the test entities is apparent by their names, the relationships aren't formed until the test functions.

Listing 125: Test Cases Setup For Divy

```
<?php

namespace HireVoice\divy-dev\Tests;
use HireVoice\Neo4J;

class DivyTests extends TestCase
{
/*******************************************************
* For testing user entites and queries related to them.
```

```
*********************************************************/
static $root;

/*********************************************************
* Users Aidan, Lauren, Nicole, and Carol for tests
*********************************************************/
static $userAidan;
static $userLauren;
static $userNicole;
static $userCarol;

/*********************************************************
* Files Apples, Luggage, and News for tests
*********************************************************/
static $fileAidansApples;
static $fileLaurensLuggage;
static $fileNicolesNews;

/*********************************************************
* For testing comments and the queries related to them.
*********************************************************/
/*********************************************************
* Comments by Aidan for tests
*********************************************************/
static $commentAidanOnAidan;
static $commentAidanOnLauren;
static $commentAidanOnNicole;
static $commentAidanOnLuggage;

/*********************************************************
* Comments by Lauren for tests
*********************************************************/
static $commentLaurenOnLauren;
static $commentLaurenOnNicole;
static $commentLaurenOnCarol;
static $commentLaurenOnNews;

/*********************************************************
* Comments by Nicole for tests
*********************************************************/
static $commentNicoleOnAidan;
static $commentNicoleOnApples;
static $commentNicoleOnLuggage;
static $commentNicoleOnNews;

/*********************************************************
```

```
* Comments by Carol for tests
*****************************************************/
static $commentCarolOnAidan;
static $commentCarolOnLauren;
static $commentCarolOnCarol;
static $commentCarolOnApples;


function setUp(){
  if(!self::$root){
  /*********************************************************
  * Generates our dates for user age and registration
  *********************************************************/
  $aidanAge=strtotime("1992 June 13");
  $laurenAge=strtotime("1992 September 14")
  $nicoleAge=strtotime("1992 September 18")
  $carolAge=strtotime("2004 September 28")
  $aidanRegister=strtotime("2013 June 1")
  $nicoleRegister=strtotime("2013 May 18")
  $laurenRegister=strtotime("2013 March 12")
  $carolRegister=strtotime("2014 May 18")

  /*********************************************************
  * Generates our dates for file upload date
  *********************************************************/
  $applesDate=strtotime("2014 May 21");
  $newsDate=strtotime("2014 May 20");
  $luggageDate=strtotime("2014 May 18");

  /*********************************************************
  * Generates our dates for comment made date and edit date
  *********************************************************/
  $commentAidanOnAidanDate = strtotime("2014 May 22");
  $commentAidanOnLaurenDate = strtotime("2014 May 22");
  $commentAidanOnNicoleDate = strtotime("2014 May 22");
  $commentAidanOnLuggageDate = strtotime("2014 May 22");

  $commentLaurenOnLaurenDate = strtotime("2014 May 23");
  $commentLaurenOnNicoleDate = strtotime("2014 May 23");
  $commentLaurenOnCarolDate = strtotime("2014 May 23");
  $commentLaurenOnNewsDate = strtotime("2014 May 23");

  $commentNicoleOnAidanDate = strtotime("2014 May 24");
  $commentNicoleOnApplesDate = strtotime("2014 May 24");
  $commentNicoleOnLuggageDate = strtotime("2014 May 24");
  $commentNicoleOnNewsDate = strtotime("2014 May 24");
```

```
$commentCarolOnAidanDate = strtotime("2014 May 25");
$commentCarolOnLaurenDate = strtotime("2014 May 25");
$commentCarolOnApplesDate = strtotime("2014 May 25");
$commentCarolOnCarolDate = strtotime("2014 May 25");

$commentEditDate = date("Y-m-d");

/*********************************************************
* Creates the tags and category arrays for the files.
*********************************************************/
$appleTags = array("apples", "trees");
$appleCategories = array("Comedy" "Drama");

$luggageTags = array("airport" , "luggage");
$luggageCategories = array("Traveling");

$newsTags = array("new", "current", "events");
$newsCategories = array("Current Events", "Technology");

/*********************************************************
* Create userAidan and give him various properties
*********************************************************/
$userAidan = new Entity\User;
$userAidan->setUserName('ACrosbie');
$userAidan->setUserId(1);
$userAidan->setUserFName('Aidan');
$userAidan->setUserLName('Crosbie');
$userAidan->setUserEmail('acrosbie@scu.edu');
$userAidan->setUserTotalOwnedFileViews(100);
$userAidan->setUserTotalOwnedFileLikes(10);
$userAidan->setUserTotalOwnedFileDislikes(50);
$userAidan->setUserTotalOwnedFilePurchases(0)
$userAidan->setUserPassword('Aidan');
$userAidan->setUserAuth(true);
$userAidan->setUserAuthCode('123456');
$userAidan->setUserDeleteAccountAuthCode('abcdef');
$userAidan->setUserCss('Standard');
$userAidan->setUserAdminLevel('divy-dev');
$userAidan->setUserAge($aidanAge);
$userAidan->setUserRegisterDate($aidanRegister);
$userAidan->setUserTotalSubscribedTo(20);
$userAidan->setUserTotalSubscribers(50);
$userAidan->setUserTotalProfileComments(10);
$userAidan->setUserTotalComments(20);
$userAidan->setUserTotalCommentLikes(50);
```

```
$userAidan->setUserTotalCommentDislikes(20);
$userAidan->setUserProfilePhoto('Aidan.jpg');

/*********************************************************
* Create userLauren and give her various properties
*********************************************************/
$userLauren = new Entity\User;
$userLauren->setUserName('LFalzarano');
$userLauren->setUserId(2);
$userLauren->setUserFName('Lauren');
$userLauren->setUserLName('Falzarano');
$userLauren->setUserEmail('lfalzarano@scu.edu');
$userLauren->setUserTotalOwnedFileViews(75);
$userLauren->setUserTotalOwnedFileLikes(20);
$userLauren->setUserTotalOwnedFileDislikes(10);
$userLauren->setUserTotalOwnedFilePurchases(10);
$userLauren->setUserPassword('Lauren');
$userLauren->setUserAuth(true);
$userLauren->setUserAuthCode('234567');
$userLauren->setUserDeleteAccountAuthCode('bcdefg');
$userLauren->setUserCss('DivyDev');
$userLauren->setUserAdminLevel('divy-dev');
$userLauren->setUserAge($laurenAge);
$userLauren->setUserRegisterDate($laurenRegister);
$userLauren->setUserTotalSubscribedTo(30);
$userLauren->setUserTotalSubscribers(40);
$userLauren->setUserTotalProfileComments(30);
$userLauren->setUserTotalComments(40);
$userLauren->setUserTotalCommentLikes(70);
$userLauren->setUserTotalCommentDislikes(60);
$userLauren->setUserProfilePhoto('Lauren.jpg');

/*********************************************************
* Create userNicole and give her various properties
*********************************************************/
$userNicole = new Entity\User;
$userNicole->setUserName('NPal')
$userNicole->setUserId(3);
$userNicole->setUserFName('Nicole')
$userNicole->setUserLName('Pal');
$userNicole->setUserEmail('npal@scu.edu');
$userNicole->setUserTotalOwnedFileViews(50);
$userNicole->setUserTotalOwnedFileLikes(20);
$userNicole->setUserTotalOwnedFileDislikes(10);
$userNicole->setUserTotalOwnedFilePurchases(0)
$userNicole->setUserPassword('Nicole');
```

```
$userNicole->setUserAuth(true);
$userNicole->setUserAuthCode('345678');
$userNicole->setUserDeleteAccountAuthCode('cdefgh');
$userNicole->setUserCss('DivyDev');
$userNicole->setUserAdminLevel('divy-dev');
$userNicole->setUserAge($nicoleAge);
$userNicole->setUserRegisterDate($nicoleRegister);
$userNicole->setUserTotalSubscribedTo(40);
$userNicole->setUserTotalSubscribers(30);
$userNicole->setUserTotalProfileComments(30);
$userNicole->setUserTotalComments(40);
$userNicole->setUserTotalCommentLikes(70);
$userNicole->setUserTotalCommentDislikes(60);
$userNicole->setUserProfilePhoto('Nicole.jpg')

/*********************************************************
* Create userCarol and give her various properties
*********************************************************/
$userCarol = new Entity\User;
$userCarol->setUserName('CCarol');
$userCarol->setUserId(4);
$userCarol->setUserFName('Carol');
$userCarol->setUserLName('Carol');
$userCarol->setUserEmail('Carol@Carol.com');
$userCarol->setUserTotalOwnedFileViews(0);
$userCarol->setUserTotalOwnedFileLikes(0);
$userCarol->setUserTotalOwnedFileDislikes(0);
$userCarol->setUserTotalOwnedFilePurchases(0);
$userCarol->setUserPassword('Carol');
$userCarol->setUserAuth(true);
$userCarol->setUserAuthCode('456789');
$userCarol->setUserDeleteAccountAuthCode('defghi');
$userCarol->setUserUserCss('Standard');
$userCarol->setUserAdminLevel('none');
$userCarol->setUserAge($carolAge);
$userCarol->setUserRegisterDate($carolRegister);
$userCarol->setUserTotalSubscribedTo(50);
$userCarol->setUserTotalSubscribers(20);
$userCarol->setUserTotalProfileComments(40);
$userCarol->setUserTotalComments(10);
$userCarol->setUserTotalCommentLikes(20);
$userCarol->setUserTotalCommentDislikes(70);
$userCarol->setUserProfilePhoto('Divy.jpg');

/*********************************************************
* Create fileAidansApples and give it various properties.
```

```
*********************************************************/
$fileAidansApples = new Entity\File;
$fileAidansApples ->setFileName('Apples');
$fileAidansApples ->setFileId(1);
$fileAidansApples ->setFileType('png');
$fileAidansApples ->setFileViews(10);
$fileAidansApples ->setFileLikes(2);
$fileAidansApples ->setFileDislikes(6);
$fileAidansApples ->setFileTags($appleTags);
$fileAidansApples ->setFileUploadDate($appleDate)
$fileAidansApples ->setFileDescr('Apples For Days')
$fileAidansApples ->setFileCategories($appleCategories);
$fileAidansApples ->setFileExplicit('Not Explicit');
$fileAidansApples ->setFilePrice(10);
$fileAidansApples ->setFileCanEditPrice(true);
$fileAidansApples ->setFileOwner($userAidan);
$fileAidansApples ->setFileTotalComments(2);
$fileAidansApples ->setFilePreview('Apples.jpg');
$fileAidansApples ->setFileIsPaid(true);

/*********************************************************
* Create fileLaurensLuggage and give it various properties.
*********************************************************/
$fileLaurensLuggage = new Entity\File;
$fileLaurensLuggage ->setFileName('Luggage');
$fileLaurensLuggage ->setFileId(2);
$fileLaurensLuggage ->setFileType('mp4');
$fileLaurensLuggage ->setFileViews(20);
$fileLaurensLuggage ->setFileLikes(4);
$fileLaurensLuggage ->setFileDislikes(3);
$fileLaurensLuggage ->setFileTags($luggageTags);
$fileLaurensLuggage ->setFileUploadDate($luggageDate);
$fileLaurensLuggage ->setFileDescr('How to pack luggage for flights')
    ;
$fileLaurensLuggage ->setFileCategories($luggageCategories);
$fileLaurensLuggage ->setFileExplicit('Not Explicit');
$fileLaurensLuggage ->setFilePrice(5);
$fileLaurensLuggage ->setFileCanEditPrice(true);
$fileLaurensLuggage ->setFileOwner($userLauren);
$fileLaurensLuggage ->setFileTotalComments(2);
$fileLaurensLuggage ->setFilePreview('Luggage.jpg');
$fileLaurensLuggage ->setFileIsPaid(true);

/*********************************************************
* Create fileNicoleNews and give it various properties.
*********************************************************/
```

```
$fileNicolesNews = new Entity\File;
$fileNicolesNews ->setFileName('Technology News For May 2014');
$fileNicolesNews ->setFileId(3);
$fileNicolesNews ->setFileType('pdf');
$fileNicolesNews ->setFileViews(30);
$fileNicolesNews ->setFileLikes(5);
$fileNicolesNews ->setFileDislikes(0);
$fileNicolesNews ->setFileTags($newsTags);
$fileNicolesNews ->setFileUploadDate($newsDate);
$fileNicolesNews ->setFileDescr('Current Events in technology');
$fileNicolesNews ->setFileCategories($newsCategories);
$fileNicolesNews ->setFileExplicit('Not Explicit');
$fileNicolesNews ->setFilePrice(0);
$fileNicolesNews ->setFileCanEditPrice(true);
$fileNicolesNews ->setFileOwner($userNicole);
$fileNicolesNews ->setFileTotalComments(25);
$fileNicolesNews ->setFilePreview('News.jpg');
$fileNicolesNews ->setFileIsPaid(false);

/*********************************************************
* Create fileDummy, which will be used in testing
*********************************************************/
$fileDummy = new Entity\File;
$fileDummy ->setFileName('Dummy File');
$fileDummy ->setFileType('png');
$fileDummy ->setFileId(4);
$fileDummy ->setFileViews(0);
$fileDummy ->setFileLikes(0);
$fileDummy ->setFileDislikes(0);
$fileDummy ->setFileTags($dummyTags);
$fileDummy ->setFileUploadDate($dummyDate);
$fileDummy ->setFileDescr('This is a dummy file to test repositories
    ');
$fileDummy ->setFileCategories($dummyCategories);
$fileDummy ->setFileExplicit('Not Explicit');
$fileDummy ->setFilePrice(0);
$fileDummy ->setFileCanEditPrice(true);
$fileDummy ->setFileOwner($userCarol);
$fileDummy ->setFileTotalComments(0);
$fileDummy ->setFilePreview('Dummy.jpg');
$fileDummy ->setFileIsPaid(false);


/*********************************************************
* Create Aidans Comment on Aidans Profile
*********************************************************/
```

```
$commentAidanOnAidan = new Entity\Comment;
$commentAidanOnAidan ->setCommentType('User');
$commentAidanOnAidan ->setCommentId(1);
$commentAidanOnAidan ->setCommentDate($commentAidanOnAidanDate);
$commentAidanOnAidan ->setCommentLikes(2);
$commentAidanOnAidan ->setCommentDislikes(0);
$commentAidanOnAidan ->setCommentUser($userAidan);
$commentAidanOnAidan ->setCommentTo($userAidan->getUserId());
$commentAidanOnAidan ->setCommentText('You are the best');
$commentAidanOnAidan ->setCommentEditDate($commentAidanOnAidanDate);
$commentAidanOnAidan ->setCommentReported(false);

/*******************************************************
* Create Aidans Comment on Laurens Profile
*******************************************************/
$commentAidanOnLauren = new Entity\Comment;
$commentAidanOnLauren ->setCommentType('User');
$commentAidanOnLauren ->setCommentId(2);
$commentAidanOnLauren ->setCommentDate($commentAidanOnLaurenDate);
$commentAidanOnLauren ->setCommentLikes(1);
$commentAidanOnLauren ->setCommentDislikes(1);
$commentAidanOnLauren ->setCommentUser($userAidan);
$commentAidanOnLauren ->setCommentTo($userLauren->getUserId());
$commentAidanOnLauren ->setCommentText('Hello');
$commentAidanOnLauren ->setCommentEditDate($commentAidanOnLaurenDate)
    ;
$commentAidanOnLauren ->setCommentReported(false);

/*******************************************************
* Create Aidans Comment on Nicoles Profile
*******************************************************/
$commentAidanOnNicole = new Entity\Comment;
$commentAidanOnNicole ->setCommentType('User');
$commentAidanOnNicole ->setCommentId(3);
$commentAidanOnNicole ->setCommentDate($commentAidanOnNicoleDate);
$commentAidanOnNicole ->setCommentLikes(1);
$commentAidanOnNicole ->setCommentDislikes(1);
$commentAidanOnNicole ->setCommentUser($userAidan);
$commentAidanOnNicole ->setCommentTo($userNicole->getUserId());
$commentAidanOnNicole ->setCommentText('Meow');
$commentAidanOnNicole ->setCommentEditDate($commentAidanOnNicoleDate)
    ;
$commentAidanOnNicole ->setCommentReported(false);

/*******************************************************
* Create Aidans Comment on the Luggage File
```

```
*******************************************************/
$commentAidanOnLuggage = new Entity\Comment;
$commentAidanOnLuggage ->setCommentType('File');
$commentAidanOnLuggage  ->setCommentId(4);
$commentAidanOnLuggage ->setCommentDate($commentAidanOnLuggageDate);
$commentAidanOnLuggage ->setCommentLikes(1);
$commentAidanOnLuggage ->setCommentDislikes(2);
$commentAidanOnLuggage ->setCommentUser($userAidan);
$commentAidanOnLuggage ->setCommentTo($fileLaurensLuggage->getFileId
    ());
$commentAidanOnLuggage ->setCommentText('Such Luggage');
$commentAidanOnLuggage ->setCommentEditDate(
    $commentAidanOnLuggageDate);
$commentAidanOnLuggage ->setCommentReported(false);

/*******************************************************
* Create Laurens Comment on Laurens Profile
*******************************************************/
$commentLaurenOnLauren = new Entity\Comment;
$commentLaurenOnLauren ->setCommentType('User');
$commentLaurenOnLauren ->setCommentId(5);
$commentLaurenOnLauren ->setCommentDate($commentLaurenOnLaurenDate);
$commentLaurenOnLauren ->setCommentLikes(2);
$commentLaurenOnLauren ->setCommentDislikes(0);
$commentLaurenOnLauren ->setCommentUser($userLauren);
$commentLaurenOnLauren ->setCommentTo($userLauren->getUserId());
$commentLaurenOnLauren ->setCommentText('I can comment on myself');
$commentLaurenOnLauren ->setCommentEditDate(
    $commentLaurenOnLaurenDate);
$commentLaurenOnLauren ->setCommentReported(false);

/*******************************************************
* Create Laurens Comment on Nicoles Profile
*******************************************************/
$commentLaurenOnNicole = new Entity\Comment;
$commentLaurenOnNicole ->setCommentType('User');
$commentLaurenOnNicole ->setCommentId(6);
$commentLaurenOnNicole ->setCommentDate($commentLaurenOnNicoleDate);
$commentLaurenOnNicole ->setCommentLikes(2);
$commentLaurenOnNicole ->setCommentDislikes(2);
$commentLaurenOnNicole ->setCommentUser($userLauren);
$commentLaurenOnNicole ->setCommentTo($userNicole->getUserId());
$commentLaurenOnNicole ->setCommentText('MeowMeowMeow');
$commentLaurenOnNicole ->setCommentEditDate(
    $commentLaurenOnNicoleDate);
$commentLaurenOnNicole ->setCommentReported(false);
```

```
/*******************************************************
* Create Laurens Comment on Carols Profile
*******************************************************/
$commentLaurenOnCarol = new Entity\Comment;
$commentLaurenOnCarol ->setCommentType('User');
$commentLaurenOnCarol ->setCommentId(7);
$commentLaurenOnCarol ->setCommentDate($commentLaurenOnLaurenDate);
$commentLaurenOnCarol ->setCommentLikes(2);
$commentLaurenOnCarol ->setCommentDislikes(1);
$commentLaurenOnCarol ->setCommentUser($userLauren);
$commentLaurenOnCarol ->setCommentTo($userCarol->getUserId());
$commentLaurenOnCarol ->setCommentText('Welcome to Divy');
$commentLaurenOnCarol ->setCommentEditDate($commentLaurenOnCarolDate)
$commentLaurenOnCarol ->setCommentReported(false);

/*******************************************************
* Create Laurens Comment on the News file
*******************************************************/
$commentLaurenOnNews = new Entity\Comment;
$commentLaurenOnNews ->setCommentType('File');
$commentLaurenOnNews ->setCommentId(8);
$commentLaurenOnNews ->setCommentDate($commentLaurenOnNewsDate);
$commentLaurenOnNews ->setCommentLikes(2);
$commentLaurenOnNews ->setCommentDislikes(1);
$commentLaurenOnNews ->setCommentUser($userLauren);
$commentLaurenOnNews ->setCommentTo($fileNicolesNews->getFileId());
$commentLaurenOnNews ->setCommentText('Very Luggage');
$commentLaurenOnNews ->setCommentEditDate($commentLaurenOnNewsDate);
$commentLaurenOnNews ->setCommentReported(false);

/*******************************************************
* Create Nicoles Comment on Aidans profile
*******************************************************/
$commentNicoleOnAidan = new Entity\Comment;
$commentNicoleOnAidan ->setCommentType('User');
$commentNicoleOnAidan ->setCommentId(9);
$commentNicoleOnAidan ->setCommentDate($commentNicoleOnAidanDate);
$commentNicoleOnAidan ->setCommentLikes(1);
$commentNicoleOnAidan ->setCommentDislikes(1);
$commentNicoleOnAidan ->setCommentUser($userNicole);
$commentNicoleOnAidan ->setCommentTo($userAidan->getUserId());
$commentNicoleOnAidan ->setCommentText('Kitten Mitens are the newest
    craze');
$commentNicoleOnAidan ->setCommentEditDate($commentNicoleOnAidanDate)
    ;
```

```
$commentNicoleOnAidan ->setCommentReported(false);

/*******************************************************
* Create Nicoles Comment on the Apples File
*******************************************************/
$commentNicoleOnApples = new Entity\Comment;
$commentNicoleOnApples ->setCommentType('File');
$commentNicoleOnApples ->setCommentId(10);
$commentNicoleOnApples ->setCommentDate($commentNicoleOnApples);
$commentNicoleOnApples ->setCommentLikes(1);
$commentNicoleOnApples ->setCommentDislikes(1);
$commentNicoleOnApples ->setCommentUser($userNicole);
$commentNicoleOnApples ->setCommentTo($fileAidansApples->getFileID())
    ;
$commentNicoleOnApples ->setCommentText('Wow');
$commentNicoleOnApples ->setCommentEditDate(
    $commentNicoleOnApplesDate);
$commentNicoleOnApples->setCommentReported(false);

/*******************************************************
* Create Nicoles Comment on the Luggage file
*******************************************************/
$commentNicoleOnLuggage = new Entity\Comment;
$commentNicoleOnLuggage ->setCommentType('File');
$commentNicoleOnLuggage ->setCommentId(11);
$commentNicoleOnLuggage ->setCommentDate($commentNicoleOnLuggageDate)
$commentNicoleOnLuggage ->setCommentLikes(1);
$commentNicoleOnLuggage ->setCommentDislikes(0);
$commentNicoleOnLuggage ->setCommentUser($userNicole);
$commentNicoleOnLuggage ->setCommentTo($fileLaurensLuggage->getFileId
    ());
$commentNicoleOnLuggage ->setCommentText('I never thought of that!');
$commentNicoleOnLuggage ->setCommentEditDate(
    $commentNicoleOnLuggageDate);
$commentNicoleOnLuggage ->setCommentReported(false);

/*******************************************************
* Create Nicoles Comment on the News file
*******************************************************/
$commentNicoleOnNews = new Entity\Comment;
$commentNicoleOnNews ->setCommentType('File');
$commentNicoleOnNews ->setCommentId(12);
$commentNicoleOnNews ->setCommentDate($commentNicoleOnNewsDate);
$commentNicoleOnNews ->setCommentLikes(1);
$commentNicoleOnNews ->setCommentDislikes(0);
$commentNicoleOnNews ->setCommentUser($userNicole);
```

```
$commentNicoleOnNews ->setCommentTo($fileNicolesNews->getFileId());
$commentNicoleOnNews ->setCommentText('Next week will feature web dev
    news');
$commentNicoleOnNews ->setCommentEditDate($commentNicoleOnNewsDate);
$commentNicoleOnNews ->setCommentReported(false);

/*******************************************************
* Create Carols Comment on Aidans profile
*******************************************************/
$commentCarolOnAidan = new Entity\Comment;
$commentCarolOnAidan ->setCommentType('User');
$commentCarolOnAidan ->setCommentId(13);
$commentCarolOnAidan ->setCommentDate($commentCarolOnAidanDate);
$commentCarolOnAidan ->setCommentLikes(0);
$commentCarolOnAidan ->setCommentDislikes(2);
$commentCarolOnAidan ->setCommentUser($userCarol);
$commentCarolOnAidan ->setCommentTo($userAidan->getUserId());
$commentCarolOnAidan ->setCommentText('You are a bad person');
$commentCarolOnAidan ->setCommentEditDate($commentCarolOnAidanDate);
$commentCarolOnAidan ->setCommentReported(false);

/*******************************************************
* Create Carols Comment on Laurens profile
*******************************************************/
$commentCarolOnLauren = new Entity\Comment;
$commentCarolOnLauren ->setCommentType('User');
$commentCarolOnLauren ->setCommentId(14);
$commentCarolOnLauren ->setCommentDate($commentCarolOnLaurenDate);
$commentCarolOnLauren ->setCommentLikes(0);
$commentCarolOnLauren ->setCommentDislikes(2);
$commentCarolOnLauren ->setCommentUser($userCarol);
$commentCarolOnLauren ->setCommentTo($userLauren->getUserId());
$commentCarolOnLauren ->setCommentText('I dont like divys style');
$commentCarolOnLauren ->setCommentEditDate($commentCarolOnLaurenDate)
    ;
$commentCarolOnLauren ->setCommentReported(false);

/*******************************************************
* Create Carols Comment on the Apples File
*******************************************************/
$commentCarolOnApples = new Entity\Comment;
$commentCarolOnApples ->setCommentType('File');
$commentCarolOnApples ->setCommentId(15);
$commentCarolOnApples ->setCommentDate($commentCarolOnApplesDate);
$commentCarolOnApples ->setCommentLikes(1);
$commentCarolOnApples ->setCommentDislikes(2);
```

```
    $commentCarolOnApples->setCommentUser($userCarol);
    $commentCarolOnApples->setCommentTo($fileAidansApples->getFileId());
    $commentCarolOnApples->setCommentText('I dont really like apples');
    $commentCarolOnApples->setCommentEditDate($commentCarolOnApplesDate)
        ;
    $commentCarolOnApples->setCommentReported(false);

    /*******************************************************
    * Create Carols Comment on Carols profile
    *******************************************************/
    $commentCarolOnCarol = new Entity\Comment;
    $commentCarolOnCarol->setCommentType('User');
    $commentCarolOnCarol->setCommentId(16);
    $commentCarolOnCarol->setCommentDate($commentCarolOnCarolDate);
    $commentCarolOnCarol->setCommentLikes(1);
    $commentCarolOnCarol->setCommentDislikes(2);
    $commentCarolOnCarol->setCommentUser($userCarol);
    $commentCarolOnCarol->setCommentTo($userCarol->getUserId());
    $commentCarolOnCarol->setCommentText('The best user on this site');
    $commentCarolOnCarol->setCommentEditDate($commentCarolOnCarolDate);
    $commentCarolOnCarol->setCommentReported(false);


    }
}


/*******************************************************
* The Test Cases Discussed in the following sections
* go here.
*******************************************************/
/*******************************************************
* 11.1.4.2 - User Entity Testing
* 11.1.4.3 - File Entity Testing
* 11.1.4.4 - Comment Entity Testing
* 11.1.4.5 - Repository Testing
*******************************************************/

}

?>
```

12.1.4.2 User Entity Testing

12.1.4.2.1 User Information Testing

Listing 126: Test User First Name Function

```
/*******************************************************
* Function: testUserFName()
*******************************************************
* Description: Tests that the get and set userFName functions
* work properly
*******************************************************/
function testUserFName(){
        $aidanFName = $userAidan->getUserFName();
        $this->assertEquals($aidanFName, 'Aidan');

        $newCarolFName = 'Crazy';
        $userCarol->setUserFName($newCarolFName);

        $carolName = $userCarol->getUserFName();
        $this->assertEquals($caronName, 'Crazy');
}
```

Listing 127: Test User Last Name Function

```
/*******************************************************
* Function: testUserLName()
*******************************************************
* Description: Test that the get and set userLName functions
* work properly
*******************************************************/
function testUserLName(){
        $nicoleLName = $userNicole->getUserLName();
        $this->assertEquals($nicoleLName, 'Pal');

        $newCarolLName = 'Meowzer';
        $userCarol->setUserLName($newCarolLName);
        $carolLName = $userCarol->getUserLName);

        $this->assertEquals($carolLName, 'Meowzer');
}
```

Listing 128: Test User Email Function

```
/*******************************************************
* Function: testUserEmail()
*******************************************************
```

```
* Description: Test that the get and set userEmail functions
* work properly
*****************************************************/
function testUserEmail(){
        $laurenEmail = $userLauren->getUserEmail();
        $this->assertEquals($laurenEmail, 'lfalzarano@scu.edu');

        $newCarolEmail ='carol@divy.com';
        $userCarol->setUserEmail($newCarolEmail);
        $this->assertEquals($newCarolEmail, 'carol@divy.com');
}
```

Listing 129: Test User Password Function

```
/*****************************************************
* Function: testUserPassword()
*****************************************************
* Description: Test that the get and set userPassword functions
* work properly
*****************************************************/
function testUserPassword(){
        $aidanMd5Password = $userAidan->getUserPassword();
        $md5Aidan = md5('Aidan');
        $this->assertEquals($md5Aidan, $aidanMd5Password);

        $carolNewPassword = md5('Meow');
        $userCarol->setUserPassword($carolNewPassword);
        $carolMd5Password = $userCarol->getUserPassword();
        $this->assertEquals($carolMd5Password, $carolNewPassword);
}
```

Listing 130: Test User Authentication Function

```
function testUserUserAuth(){
        $laurenIsAuth = $userLauren->getUserAuth();
        $this->assertTrue($laurenIsAuth);

        $newCarolAuth = false;
        $userCarol->setUserAuth($newCarolAuth);
        $carolIsAuth = $userCarol->getUserAuth();
        $this->assertFalse($carolIsAuth);
}
```

Listing 131: Test User Authentication Code Function

```
/*************************************************************
* Function: testUserAuthCode()
*************************************************************
* Description: Test that the get and set userAuthCode functions
* work properly
*************************************************************/
function testUserAuthCode(){
        $nicoleAuthCode = $userNicole->getUserAuthCode();
        $this->assertEquals($nicoleAuthCode, '345678');

        $newCarolAuthCode = '012345';
        $userCarol->setUserAuthCode($newCarolAuthCode);
        $carolAuthCode = $userCarol->getUserAuthCode();
        $this->assertEquals($newCarolAuthCode, $carolAuthCode);
}
```

Listing 132: Test User Delete Account Authentication Code Function

```
/*************************************************************
* Function: testUserDeleteAccountAuthCode()
*************************************************************
* Description: Test that the get and set userDeleteAccountAuthCode
* functions work properly
*************************************************************/
function testUserDeleteAccountAuthCode(){
        $aidanDeleteAccountAuthCode = $userAidan->
            getUserDeleteAccountAuthCode();
        $this->assertEquals($aidanDeleteAccountAuthCode, 'abcdef');

        $newCarolDeleteAccountCode = 'asdfgh';
        $userCarol->setUserDeleteAccountAuthCode(
            $newCarolDeleteAccountCode);
        $carolDeleteAccountCode = $userCarol->
            getUserDeleteAccountAuthCode();
        $this->assertEquals($newCaroleDeleteAccountCode,
            $carolDeleteAccountCode);
}
```

Listing 133: Test User Age Function

```
/*********************************************************
* Function: testUserAge()
*********************************************************
* Description: Test that the get and set userAge functions
* work properly
*********************************************************/
function testUserAge(){
        $aidanAge = $userAidan->getUserAge();
        $aidanActualAge = strtotime("1992 June 13");
        $this->assertEquals($aidanAge, $aidanActualAge);

        $carolNewAge = date("Y-m-d");
        $userCarol->setUserAge($carolNewAge);
        $carolAge = $userCarol->getUserAge();
        $this->assertEquals($carolNewAge, $carolAge);
}
```

Listing 134: Test User Register Date Function

```
/*********************************************************
* Function: testUserRegisterDate()
*********************************************************
* Description: Test that the get and set userRegisterDate
* functions work properly
*********************************************************/
function testUserRegisterDate(){
        $laurenRegisterDate = $userLauren->getUserRegisterDate();
        $laurenActualRegisterDate = strtotime("2013 March 12");
        $this->assertEquals($laurenRegisterDate,
            $laurenActualRegisterDate);

        $carolNewRegisterDate = date("Y-m-d");
        $userCarol->setUserRegisterDate($carolNewRegisterDate);
        $carolRegisterDate = $userCarol->getUserRegisterDate();
        $this->assertEquals($carolNewRegisterDate, $carolRegisterDate);
}
```

12.1.4.2.2 User Profile Information Testing

Listing 135: Test User Css Function

```
/*************************************************
* Function: testUserCss()
*************************************************
* Description: Test that the get and set userCss functions
* work properly
*************************************************/
function testUserCss(){
        $nicoleCss = $userNicole->getUserCss();
        $this->assertEquals($nicoleCss, 'DivyDev');

        $newCarolCss = 'SuperSecret';
        $userCarol->setUserCss($newCarolCss);
        $carolCss = $userCarol->getUserCss();
        $this->assertEquals($newCarolCss, $carolCss);
}
```

Listing 136: Test User Profile Photo Function

```
/*************************************************
* Function: testUserProfilePhoto()
*************************************************
* Description: Test that the get and set userProfilePhoto
* functions work properly
*************************************************/
function testUserProfilePhoto(){
        $laurenPhoto = $userLauren->getUserProfilePhoto()
        $this->assertEquals($laurenPhoto, 'Lauren.jpg');

        $carolNewPhoto = 'Carol.jpg';
        $userCarol->setUserProfilePhoto($carolnewPhoto);
        $carolPhoto = $userCarol->getUserProfilePhoto();
        $this->assertEquals($carolNewPhoto, $carolPhoto);
}
```

Listing 137: Test User Admin Level Function

```
/*************************************************
* Function: testUserAdminLevel()
*************************************************
* Description: Test that the get and set userAdminLevel
* functions work properly
*************************************************/
```

```
function testUserAdminLevel(){
        $laurenAdminLevel = $userLauren->getUserAdminLevel();
        $this->assertEquals($laurenAdminLevel, 'divy-dev')

        $newCarolAdminLevel = 'Super User';
        $userCarol->setUserAdminLevel($newCarolAdminLevel);
        $carolAdminLevel = $userCarol->getUserAdminLevel();
        $this->assertEquals($carolAdminLevel, $newCarolAdminLevel);
}
```

12.1.4.2.3 User Interaction Testing

Listing 138: Test User SubscribedTo/Subscribers Function

```
/*********************************************************
* Function: testUserSubscribeTo()
*********************************************************
* Description: Test all the aspects of adding removing an
* existing subscription/subscriber.
*********************************************************/
function testUserSubscribedTo(){
        $userCarol->addUserSubscribedTo($userAidan);
        $userCarol->addUserSubscribedTo($userNicole);
        $userCarol->addUserSubscribedTo($userLaruen);
        $userAidan->addUserSubscribers($userCarol);
        $userNicole->addUserSubscribers($userCarol);
        $userLauren->addUserSubscribers($userCarol);

        $userCarol->removeUserSubscribedTo($userAidan);
        $userCarol->removeUserSubscribedTo($userLaruen);
        $userAidan->removeUserSubscribers($userCarol);
        $userLauren->removeUserSubscribers($userCarol);

        $aidanSubscribers = $userAidan->getUserSubscribers();
        $nicoleSubscribers = $userNicole->getUserSubscribers();
        $laurenSubscribers = $userLauren->getUserSubscribers();

        $carolsSubscribedTo = $userCarol->getUserSubscribedTo();

        $carolSubscribeToAidan = $userAidan->hasUserSubscribers(
            $userCarol);
        $carolSubscribeToNicole = $userNicole->hasUserSubscribers(
            $userCarol);
```

```
        $carolSubscribedToLauren = $userLaruen->hasUserSubscribed(
            $userCarol);

        $aidanSubscriberCarol = $userCarol->hasUserSubscribedTo(
            $userAidan);
        $nicoleSubscriberCarol = $userCarol->hasUserSubscribedTo(
            $userNicole);
        $laurenSubscriberCarol = $userCarol->hasUserSubscribedTo(
            $userLauren);

        $this->assertTrue($carolSubscribeToNicole);
        $this->assertTrue($nicoleSubscriberCarol);
        $this->assertFalse($carolSubscribeToAidan);
        $this->assertFalse($aidanSubscriberCarol);
        $this->assertFalse($carolSubscribedToLauren);
        $this->assertFalse($laurenSubscriberCarol);
}
```

Listing 139: Test User Profile Comments Function

```
/*******************************************************
* Function: testUserProfileComments()
*******************************************************
* Description: Test the aspects of adding and removing a
* new profile comment
*******************************************************/
function testUserProfileComments(){
        $userAidan->addUserProfileComments($commentCarolOnAidan);
        $userAidan->addUserprofileComments($commentNicoleOnAidan);
        $userAidan->addUserProfileComments($commentAidanOnAidan);

        $userAidan->removeUserProfileComments($commentAidanOnAidan);

        $commentsOnAidan = $userAidan->getUserProfileComments();
        $userAidan->setUserProfileComments($commentsOnAidan);

        $didAidanComment = $userAidan->hasUserProfileComments(
            $userAidan);
        $didNicoleComment = $userAidan->hasUserProfileComments(
            $userNicole);
        $didLaurenComment = $userAidan->hasUserProfileComments(
            $userLauren);

        $this->assertTrue($didNicoleComment);
```

```
        $this->assertTrue($didLaurenComment);
        $this->assertFalse($didAidanComment);
}
```

12.1.4.3 File Entity Testing

12.1.4.2.3.2 Test User Proile Comments

12.1.4.3.1 File Information Testing

Listing 140: Test File Owner Function

```
/*******************************************************
* Function: testFileOwner()
*******************************************************
* Description: Test the get and set functions of the file
* owner property
*******************************************************/
function testFileOwner(){
        $applesOwner = $fileAidansApples->getFileOwner();
        $appleOwnerName = $appleOwner->getUserName();
        $aidanName = $userAidan->getUserName();
        $this->assertEquals($appleOwnerName, $aidanName);

        $newNewsOwner = $userAidan;
        $fileNicolesNews->setFileOwner($newNewsOwner);
        $newsOwner = $fileNicolesNews->getFileOwner();
        $newsOwnerName = $newsOwner->getUserName();
        $this->assertEquals($aidanName, $newsOwnerName);
}
```

Listing 141: Test File Name Function

```
/*******************************************************
* Function: testFileName()
*******************************************************
* Description: Test the get and set functions of the file
* name property
*******************************************************/
function testFileName(){
        $appleFileName = $fileAidansApples->getFileName();
        $this->assetEquals($appleFileName, 'Apples');
```

```
        $newLuggageFileName = 'Packing Your Suitcase';
        $fileLaurensLuggage->setFileName($newLuggageFileName);
        $luggageFileName = $fileLaurensLuggage->getFileName();
        $this->assertEquals($luggageFileName, $newLuggageFileName);
}
```

Listing 142: Test File Type Function

```
/*******************************************************
* Function: testFileType()
*******************************************************
* Description: Test the get and set functions of the file
* type property
*******************************************************/
function testFileType(){
        $newsFileType = $fileNicolesNews->getFileType();
        $this->assertEquals($newsFileType, 'pdf');

        $newApplesFileType = 'jpg';
        $fileAidansApples->setFileType($newApplesFileType);
        $applesFileType = $fileAidansApples->getFileType();
        $this->assertEquals($applesFileType, $newApplesFileType);
}
```

Listing 143: Test File Tags Function

```
/*******************************************************
* Function: testFileTags()
*******************************************************
* Description: Test the get,set, and add function of the
* file tags property
*******************************************************/
function testFileTags(){
        array $luggageFileTags = $fileLaurensLuggage->getFileTags();
        $this->assertEquals($luggageFileTags[0], 'airport');

        $applesNewTags = array ("WackyApples");
        $fileAidansApples->setFileTags($applesNewTags);
        $applesTags = $fileAidansApples->getFileTags);
        $this->assertEquals($appleTags[0], 'WacklyApples');
}
```

Listing 144: Test File Categories Function

```
/*************************************************
* Function: testFileCategories()
*************************************************
* Description: Test the get, set, and add functions of the
* file categories property
*************************************************/
function testFileCategories(){
        array $newsCategories = $fileNicolesNews->getFileCategories();
        $this->assertEquals($newsCategories[0], "Current Events");

        $newLuggageCategories = array("How To" , "Traveling");
        $fileLaurensLuggage->setFileCategories($newLuggageCategories);
        array $luggageCategories = $fileLaurensLuggage->
            getFileCategories();
        $this->assertEquals($luggageCategories[0], "How To");
        $this->assertEquals($luggageCategories[1], "Traveling");
}
```

Listing 145: Test File Description Function

```
/*************************************************
* Function: testFileDescr()
*************************************************
* Description: Test the get and set functions of the file
* description property
*************************************************/
function testFileDescr(){
        $applesDescr = $fileAidansApples->getFileDescr();
        $this->assertEquals($appleDescr, 'Apples For Days');

        $newNewsDescr = 'Technology News for May 21st';
        $fileNicolesNews->setFileDescr($newNewsDescr);
        $newsDescr = $fileNicolesNews->getFileDescr();
        $this->assertEquals($newNewsDescr, $newsDescr);
}
```

Listing 146: Test File Explicit Function

```
/*************************************************
* Function: testFileExplicit()
*************************************************
```

```
* Description: Test the get and set functions of the file
* explicit property
*******************************************************/
function testFileExplicit(){
        $areApplesExplicit = $fileAidansApples->getFileExplicit();
        $this->assertEquals($areApplesExplicit, 'Not Explicit');

        $newLuggageExplicit = 'Explicit';
        $fileLaurensLuggage->setFileExplicit($newLuggageExplicit);
        $luggageExplicit = $fileLaurensLuggage->getFileExplicit();
        $this->assertEquals($luggageExplicit, 'Explicit');
}
```

Listing 147: Test File Price Function

```
/*******************************************************
* Function: testFilePrice()
*******************************************************
* Description: Tests the aspects of setting a new file price
* This includes updating fileCanEditPrice and fileIsPaid
*******************************************************/
function testFilePrice(){
        $applesPrice = $fileAidansApples->getFilePrice();
        $this->assertEquals($applesPrice, 10);

        $newNewsPrice = 5;
        $newNewsIsPaid = 'Paid';
        $newNewsCanEditPrice = true;
        $fileNicolesNews->setFilePrice($newNewsPrice);
        $fileNicolesNews->setFileIsPaid($newNewsIsPaid);
        $fileNicolesNews->setFileCanEditPrice($newNewsCanEditPrice);

        $newLuggageprice = 0;
        $newLuggageIsPaid = 'Free';
        $newLuggageCanEditPrice = false;
        $fileLaurensLuggage->setFilePrice($newLuggagePrice);
        $fileLaurensLuggage->setFileIsPaid($newLuggageIsPaid);
        $fileLaurensLuggage->setFileCanEditPrice(
            $newLuggageCanEditPrice);

        $newsPrice = $fileNicolesNews->getFilePrice();
        $newsIsPaid = $fileNicolesNews->getFileIsPaid();
        $newsCanEditPrice = $fileNicolesNews->getFileCanEditPrice();
```

```
        $luggagePrice = $fileLaurensLuggage->getFilePrice();
        $luggageIsPaid = $fileLaurensLuggage->getFileIsPaid();
        $luggageCanEditPrice = $fileLaurensLuggage->getFileCanEditPrice
            ();

        $this->assertEquals($newsPrice, 5);
        $this->assertEquals($newsIsPaid, 'Paid');
        $this->assertTrue($newsCanEditPrice);

        $this->assertEquals($luggagePrice, 0);
        $this->assertEquals($luggageIsPaid, 'Free');
        $this->assertFalse($luggageCanEditPrice);
}
```

12.1.4.3.2 File Interactions Testing

Listing 148: Test File Collaborators Function

```
/*******************************************************
* File Interaction Testing
*******************************************************/
/*******************************************************
* Function: testFileCollaborators()
*******************************************************
* Description: Tests the aspects of adding and removing
* file collaborators
*******************************************************/
function testFileCollaborators(){
        $fileAidansApples->addFileCollaborators($userNicole);
        $fileAidansApples->addFileCollaborators($userLauren);
        $fileAidansApples->addFileCollaborators($userCarol);

        $userNicole->addUserCollaboratedFiles($fileAidansApples);
        $userLauren->addUserCollaboratedFiles($fileAidansApples);
        $userCarol->addUserCollaboratedFiles($fileAidansApples);

        $fileAidansApples->removeFileCollaborators($userCarol);
        $userCarol->removeUserCollaboratedFiles($fileAidansApples);

        $applesCollaborators = $fileAidansApples->getFileCollaborators
            ();
        $nicolesCollaboratedFiles = $userNicole->
            getUserCollaboratedFiles();
```

```
        $laurensCollaboratedFiles = $userLauren->
            getUserCollaboratedFiles();
        $carolsCollaboratedFiles = $userCarol->getUserCollaboratedFiles
            ();

        $fileAidansApples->setFileCollaborators($applesCollaborators);
        $userNicole->setUserCollaboratedFiles($nicolesCollaboratedFiles
            );
        $userLauren->setUserCollaboratedFiles($laurensCollaboratedFiles
            );
        $userCarol->setUserCollaboratedFiles($carolsCollaboratedFiles);

        $applesHasCarol = $fileAidansApples->hasFileCollaborators(
            $userCarol);
        $carolHasApples = $userCarol->hasUserCollaboratedFiles(
            $fileAidansApples);

        $applesHasNicole = $fileAidansApples->hasFileCollaborators(
            $userNicole);
        $nicoleHasApples = $userNicole->hasUserCollaboratedFiles(
            $fileAidansApples);

        $applesHasLauren = $fileAidansApples->hasFileCollaborators(
            $userLaruen);
        $laurenHasApples = $userLauren->hasUserCollaboratedFiles(
            $fileAidansApples);

        $this->assertFalse($applesHasCarol);
        $this->assertFalse($carolHasApples);

        $this->assertTrue($applesHasNicole);
        $this->assertTrue($nicoleHasApples);
        $this->assertTrue($applesHasLauren);
        $this->assertTrue($laurenHasApples);

}
```

Listing 149: Test File Comments Function

```
/*******************************************************
* Function: testFileComments()
*******************************************************
* Description: Tests the aspects of adding and removing
* file comments
```

```
*******************************************************/
function testFileComments(){
        $fileNicolesNews->addFileComments($commentAidanOnNews);
        $fileNicolesNews->addFileComemnts($commentNicoleOnNews);
        $userAidan->addUserFileComments($commentAidanOnNews);
        $userNicole->addUserFileComments($commentNicoleOnNews);

        $fileNicoleOnNews->removeFileComments($commentNicoleOnNews);
        $userNicole->removeUserFileComments($fileNicoleOnNews);

        $nicolesNewsComments = $fileNicolesNews->getFileComments();
        $fileNicoleNews->setFileComments($nicolesNewsComments);

        $aidansComments = $userAidan->getUserFileComments();
        $userAidan->setUserFileComments($aidansComments);
        $nicolesComments = $userNicole->getUserFileComments();
        $userNicole->setUserFileComments($nicolesComments);

        $didAidanComment = $fileNicolesNews->hasFileComments(
            $commentAidanOnNews);
        $didNicoleComment = $fileNicolesNews->hasFileComments(
            $commentNicoleOnNews);
        $didSaveAidanComment = $userAidan->hasUserFileComments(
            $commentAidanOnNews);
        $didSaveNicoleComment = $userNicole->hasUserFileComments(
            $commentNicoleOnNews);

        $this->assertTrue($didAidanComment);
        $this->assertTrue($didSaveAidanComment);
        $this->assertFalse($didNicoleComment);
        $this->assertFalse($didSaveNicoleComment);
}
```

Listing 150: Test File Views Function

```
/*******************************************************
* Function: testFileUserViews()
*******************************************************
* Description: Tests the aspects of adding and removing a
* file view
*******************************************************/
function testFileViews(){
        $fileAidansApples->addFileUserViews($userLauren);
        $fileAidansApples->addFileUserViews($userCarol);
```

```
        $userLauren->addUserFileViews($fileAidansApples);
        $userCarol->addUserFileViews($fileAidansApples);

        $fileAidansApples->removeFileUserViews($userCarol);
        $userCarol->removeUserFileViews($fileAidansApples);

        $applesViews = $fileAidansApples->getFileUserViews();
        $userLaurenViewed = $userLauren->getUserFileViews();
        $userCarolViewed = $userCarol->getUserFileViews();

        $fileAidansApples->setFileUserViews($appleViews);
        $userLauren->setUserFileViews($userLaurenViewed);
        $userCarol->setUserFileViews($userCarolViewed);

        $doesApplesStoreCarolView = $fileAidansApples->hasFileUserViews
            ($userCarol);
        $doesCarolStoreApplesView = $userCarol->hasUserFileViews(
            $fileAidansApples);
        $doesApplesStoreLaurenView = $fileAidansApples->
            hasFileUserViews($userLauren);
        $doesLaurenStoreApplesView = $userLauren->hasUserFileViews(
            $fileAidansApples);

        $this->assertTrue($doesApplesStoreLaurenView);
        $this->assertTrue($doesLaurenStoreAppleView);
        $this->assertFalse($doesApplesStoreCarolView);
        $this->assertFalse($doesCarolStoreApplesView);
}
```

Listing 151: Test File Likes Function

```
/*******************************************************
* Function: testFileUserLikes()
*******************************************************
* Description: Tests the aspects of adding and removing a
* file like
*******************************************************/
function testFileUserLikes(){
        $fileLaurensLuggage->addFileUserLikes($userNicole);
        $fileLaurensLuggage->addFileUserLikes($userLauren);
        $userNicole->addUserFileLikes($fileLaurensLuggage);
        $userLauren->addUserFileLikes($fileLaurensLuggage);
```

```
        $fileLaurensLuggage->removeFileUserLikes($userNicole);
        $userNicole->removeUserFileLikes($fileLaurensLuggage);

        $luggageLikes = $fileLaurensLuggage->getFileUserLikes();
        $laurenLikes = $userLaruen->getUserFileLikes();
        $nicoleLikes = $userNicole->getUserFileLikes();

        $fileLaurensLuggage->setFileUserLikes($luggageLikes);
        $userLauren->setUserFileLikes($laurenLikes);
        $userNicole->setUserFileLikes($nicoleLikes);

        $luggageStoreLView = $fileLaurensLuggage->hasFileUserLikes(
            $userLauren);
        $laurenStoreLView = $userLauren->hasUserFileLikes(
            $fileLaurensLuggage);
        $luggageStoreNView = $fileLaurensLuggage->hasFileUserLikes(
            $userNicole);
        $nicoleStoreLView = $userNicole->hasUserFileLikes(
            $fileLaurensLuggage);

        $this->assertTrue($luggageStoreLView);
        $this->assertTrue($laurenStoreLView);
        $this->assertFalse($luggageStoreNView;
        $this->assertFalse($nicoleStoreLView);
}
```

Listing 152: Test File Dislikes Function

```
/*******************************************************
* Function: testFileUserDislikes()
*******************************************************
* Description: Tests the aspects of adding and removing a
* file dislike
*******************************************************/
function testFileUserDislikes(){
        $fileAidansApples->addFileUserDislikes($userCarol);
        $fileAidansApples->addFileUserDislikes($userNicole);
        $userCarol->addUserFileDislikes($fileAidansApples);
        $userNicole->addUserFileDislikes($fileAidansApples);

        $fileAidansApples->removeFileUserDislikes($userCarol);
        $userCarol->removeUserFileDislikes($fileAidansApples);

        $applesDislikes = $fileAidansApples->getFileUserDislikes();
```

```
        $carolDislikes = $userCarol->getUserFileDislikes();
        $nicoleDislikes = $userNicole->getUserFileDislikes();

        $fileAidansApples->setFileUserDislikes($applesDislikes);
        $userCarol->setFileUserDislikes($carolDislikes);
        $userNicole->setFileUserDislikes($nicoleDislikes);

        $applesStoreCarolDislike = $fileAidansApples->
            hasFileUserDislikes($userCarol);
        $applesStoreNicoleDislike = $fileAidansApples->
            hasFileUserDislikes($userNicole);
        $carolStoreApplesDislike = $userCarol->hasUserFileDislikes(
            $fileAidansApples);
        $nicoleStoreApplesDislike = $userNicole->hasUserFileDislikes(
            $fileAidansApples);

        $this->assertTrue($applesStoreNicoleDislike);
        $this->assertTrue($nicoleStoreApplesDislike);
        $this->assertFalse($applesStoreCarolDislike);
        $this->assertFalse($carolStoreApplesDislike);
}
```

Listing 153: Test File Purchases Function

```
/*******************************************************
* Function: testFileUserPurchases()
*******************************************************
* Description: Tests the aspects of adding and removing
* a file purchase
*******************************************************/
function testFile(){
        $fileLaurensLuggage->addFileUserPurchases($userCarol);
        $fileLaurensLuggage->addFileUserPurchases($userCarol);
        $userCarol->addUserFilePurchases($fileLaurensLuggage);
        $userNicole->addUserFilePurchases($fileLaurensLuggage);

        $fileLaurensLuggage->removeFileUserPurchases($userCarol);
        $userCarol->removeUserFilePurchases($fileLaurensLuggage);

        $filePurchases = $fileLaurensLuggage->getFileUserPurchases();
        $userCarol = $userCarol->getUserFilePurchases();
        $userNicole = $userNicole->getUserFilePurchases();
```

```
        $luggageStoreNicolePurchase = $fileLaurensLuggage->
            hasFileUserPurchases($userNicole);
        $luggageStoreCarolPurchase = $fileLaurensLuggage->
            hasFileUserPurchases($userCarol);
        $userCarolStoreLuggagePurchase = $userCarol->
            hasUserFilePurchases($fileLaurensLuggage);
        $userNicoleStoreLuggagePurchase = $userNicole->
            hasUserFilePurchases($fileLaurensLuggage);

        $this->assertTrue($luggageStoreNicolePurchase);
        $this->assertTrue($userNicoleStoreLuggagePurchase);
        $this->assertFalse($luggageStoreCarolPurchase);
        $this->assertFalse($userCarolStoreLuggagePurchase);
}
```

12.1.4.4 Comment Entity Testing

12.1.4.3.2.6 Test File User Purchases

12.1.4.4.1 Comment Interaction Testing

Listing 154: Test Comment Text Function

```
/*********************************************************
* Function: testCommentText()
*********************************************************
* Description: Tests the get and set functions of the
* comment text.
*********************************************************/
function testCommentText(){
        $carolApplesText = $commentCarolOnApples->getCommentText();
        $nicoleApplesText = $commentNicoleOnApples->getCommentText();

        $this->assertEquals($carolApplesText, 'I dont really like
            apples';
        $this->assertEquals($nicoleApplesText, 'Wow')

        $carolNewText = 'I love apples';
        $carolNewEditDate = date("Y-m-d");
        $commentCarolOnApples->setCommentText($commentNewText);
        $commentCarolOnApples->setCommentEditDate($carolNewEditDate);

        $carolEditDate = $commentCarolOnApples->getCommentEditDate();
        $carolCommentText = $commentCarolOnApples->getCommentText();
```

```
        $this->assertEquals($carolEditDate, $carolNewText);
        $this->assertEquals($carolCommentText, $carolNewEditDate);
}
```

Listing 155: Test Comment Likes By Users Function

```
/*******************************************************
* Function: testCommentLikes()
*******************************************************
* Description: Tests the aspects of adding and removing
* a comment like
*******************************************************/
function testCommentLikes(){
        $commentAidanOnNews->addCommentLikesByUsers($userAidan);
        $commentAidanOnNews->addCommentLikesByUsers($userLauren);

        $commentAidanOnNews->removeCommentLikesByUsers($userLauren);

        $commentLikesByUsers = $commentAidanOnNews->
            getCommentLikesByUsers();

        $commentAidanOnNews->setCommentLikesByUsers(
            $commentLikesByUsers);

        $commentStoreAidanLike = $commentAidanOnNews->
            hasCommentLikesByUsers($userAidan);
        $commentStoreLaurenLike = $commentAidanOnNews->
            hasCommentLikesByUsers($userLauren);

        $this->assertTrue($commentStoreAidanLike);
        $this->assertFalse($commentStoreLaurenLike);
}
```

12.1.4.5 Repository Testing

12.1.4.4.1.2 Test Comment Likes By Users For clarity sake, cypher queries that are independent of recommending based on user relationships to other users and files will have preset total counts. These total counts are evaluated at the time when the relationship is made. However, for ease of testing these values have been preset. Functions such as findMostSubscribedToUsersAfterSubscribedTo will be based on the number of relationships formed. Ids are normally automatically generated, but the sake of testing repository queries, these have been preset as well.

12.1.4.5.1 User Repository Testing

Listing 156: Test User Repository Find One User By Id Function

```
/*******************************************************
* Function: testUserRepoFindOneUserById()
*******************************************************
* Description: Tests that finding a single user by the
* passed Id works properly.
*******************************************************/
function testUserRepoFindOneUserById(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $aidanId = $userAidan->getUserId();
        $checkUser = $userRepo->findOneUserById($aidanId);
        $checkUserName = $checkUser->getUserName();
        $aidanName = $userAidan->getUserName();

        $this->assertEquals($checkUserName, $aidanName);
}
```

Listing 157: Test User Repository Find One User By Name Function

```
/*******************************************************
* Function: testUserRepoFindOneUserByName()
*******************************************************
* Description: Tests that finding a single user by the
* name passed works properly
*******************************************************/
function testUserRepoFiindOneUserByName(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $nicoleName = 'NPal';
        $checkUser = $userRepo->findOneUserByName($nicoleName);
        $checkUserName = $checkUser->getUserName();

        $this->assertEquals($checkUserName, $nicoleName);
}
```

Listing 158: Test User Repository Find One User By Email Function

```
/*******************************************************
* Function: testUserRepoFindOneUserByEmail()
```

```
*******************************************************
* Description: Tests that finding a single user by their
* email works properly
*******************************************************/
function testUserRepoFindOneUserByEmail(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $laurenName = $userLauren->getUserName();
        $laurenEmail = $userLauren->getUserEmail();
        $checkUser = $userRepo->findOneUserByEmail($laurenEmail);
        $checkUserName = $checkUser->getUserName();

        $this->assertEquals($checkUserName, $laurenName);
}
```

Listing 159: Test User Repository Find Most Relevant Users Function

```
/*******************************************************
* Function: testUserRepoFindMostRelevantUsers()
*******************************************************
* Description: Tests that the list of relevant users
* returned is correct
*******************************************************/
function testUserRepoFindMostRelevantUsers(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $searchName = $userAidan->getUserName();
        $nameSimilarToAidan = 'ACrowbar';
        $userCarol->setUserName($nameSimilarToAidan);

        $relevantUsers = $userRepo->findMostRelevantUsers($searchName);
        $this->assertCount(2, count($relevantUsers);

        $firstResult = $relevantUsers->first();
        $this->assertEquals($firstResult->getUserName, $searchName);
}
```

Listing 160: Test User Repository Find Most Subscribed To Function

```
/*******************************************************
```

```
* Function: testUserRepoFindMostSubscribedToUsers()
*********************************************************
* Description: Tests that the list of most subscribed to
* users returned is correct
*********************************************************/
function testUserRepoFindMostSubscribedToUsers(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $searchName = $userLauren->getUserName();
        $nameSimilarToLauren = 'LFanOfDivy';
        $userCarol->setUserName($nameSimilarToLauren);

        $relevantUsersOrderedBySubscribers = $userRepo->
            findMostSubscribedToUsers($searchName);
        $this->assertCount(2, count($relevantUsersOrderedBySubscribers)
            );

        $firstInResult = $relevantUsersOrderedBySubscribers->first();
        $this->assertEquals($firstInResult->getUserName(), $searchName)
}
```

Listing 161: Test User Repository Find Users With Most File Views Function

```
/*********************************************************
* Function: testUserRepoFindUsersWithMostFileViews()
*********************************************************
* Description: Tests the returned list of users with the
* the most file views is correct
*********************************************************/
function testUserRepoFindUsersWithMostFileViews(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $searchName = $userNicole->getUserName();
        $nameSimilarToNicole = 'NPan';
        $userCarol->setUserName($nameSimilarToNicole);

        $relevantUsersOrderedByFileViews = $userRepo->
            findUsersWithMostFileViews($searchName);
        $this->assertCount(2, count($relevantUsersOrderedByFileViews));

        $firstInResult = $relevantUsersOrderedByFileViews->first();
```

```
        $this->assertEquals($firstInResult->getUserName(), $searchName)
            ;
}
```

Listing 162: Test User Repository Find Users With Most File Likes Function

```
/*******************************************************
* Function: testUserRepoFindUsersWithMostFileLikes()
*******************************************************
* Description: Tests the returned list of users with the
* most file likes is correct
*******************************************************/
function testUserRepoFindUsersWithMostFileLikes(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $searchName = $userAidan->getUserName();
        $nameSimilarToAidan 'ACrow';
        $userNicole->setUserName($nameSimilarToAidan);

        $relevantUsersOrderedByFileLikes = $userRepo->
            findUsersWithMostFileLikes($searchName);
        $this->assertCount(2, count($relevantUsersOrderedByFileViews));

        $firstInResult = $relevantUsersOrderedByFileViews->first();
        $this->assertEquals($firstInResult->getUserName,
            $nameSimilarToAidan);
}
```

Listing 163: Test User Repository Find Newest Users Function

```
/*******************************************************
* Function: testUserRepoFindNewestUsers()
*******************************************************
* Description: Tests that the returned list of newest users
* is correct
*******************************************************/
function testUserRepoFindNewestUsers(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $searchName = $userLauren->getUserName();
```

```
        $nameSimilarToLauren = 'LFanOfDivy';
        $userAidan->setUserName($nameSimilarToLauren);

        $newestRelevantUsers = $userRepo->findNewestUsers($searchName);
        $this->assertCount(2, count($newestRelevantUsers));

        $firstInResult = $newestRelevantUsers->first();
        $this->assertEquals($firstInResult->getUserName(),
            $nameSimilarToLauren);
}
```

Listing 164: Test User Repository Find Users With Most Comment Likes

```
/*******************************************************
* Function: testUserRepoFindUsersWithMostCommentLikes()
*******************************************************
* Description: Tests that the list of users with the most
* comment likes is returned properly
*******************************************************/
function testUserRepoFindUsersWithMostCommentLikes(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $searchName = $userCarol->getUserName();
        $nameSimilarToCarol = 'CarManiac';
        $userAidan->setUserName($nameSimilarToCarol);

        $relevantUsersOrderedByCommentLikes = $userRepo->
            findUsersWithMostCommentLikes($searchName);
        $this->assertCount(2, count($relevantUsersOrderedByCommentLikes
            ));

        $firstInResult = $relevantUsersOrderedByCommentLikes->first();
        $this->assertEquals($firstInResultName->getUserName(),
            $nameSimilarToCarol);
}
```

Listing 165: Test User Repository Find Most Subscribed To Users After Subscribed To

```
/*******************************************************
* Function: testUserRepoFindMostSubscribedToUsersAfterSubscribedTo()
*******************************************************
```

```
* Description: Tests that the returned list of most
* subscribed to users based on users who also subscribed to
* is correct
*****************************************************/
function testUserRepoFindMostSubscribedToUsersAfterSubscribedTo(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $userAidan->addUserSubscribedTo($userCarol);
        $userAidan->addUserSubscribedTo($userNicole);
        $userAidan->addUserSubscribedTo($userLauren);
        $userNicole->addUserSubscribedTo($userCarol);
        $userLauren->addUserSubscribedTo($userNicole);
        $userLauren->addUserSubscribedTo($userCarol);

        $mostSubscribedToAfterCarol = $userRepo->
            findMostSubscribedToAfterSubscribedTo($userCarol);

        $this->assertCount(2, count($mostSubscribedToAfterCarol));
        $firstResult = $mostSubscribedToAfterCarol;
        $this->assertEquals($firstResult->getUserName, $userNicole->
            getUserName());
}
```

Listing 166: Test User Repository Find Most Subscribed To Users With File Like

```
/*****************************************************
* Function: testUserRepoFindMostSubscribedToUsersWithFileLike()
*****************************************************
* Description: Tests that the returned list of most subscribed
* to users based on users who like the file is correct.
*****************************************************/
function testUserRepoFindMostSubscribedToUsersWithFileLike(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $fileAidansApples->addFileUserLikes($userNicole);
        $fileAidansApples->addFileUserLikes($userCarol);
        $fileAidansApples->addFileUserLikes($userLauren);

        $userNicole->addUserSubscribedTo($userAidan);
        $userNicole->addUserSubscribedTo($userLauren);
        $userNicole->addUserSubscribedTo($userCarol);
```

```
        $userLauren->addUserSubscribedTo($userAidan);
        $userLauren->addUserSubscribedTo($userCarol);

        $userLauren->addUserSubscribedTo($userAidan);

        $mostSubscribedToAfterLikingApples = $userRepo->
            findMostSubscribedToUsersWithFileLike($fileAidansApples);

        $this->assertCount(3, count($mostSubscribedToAfterLikingApples)
            );
        $mostSubscribedTo = $mostSubscribedToAfterLikingApples->first()
            ;
        $this->assertEquals($userAidan->getName(), $mostSubscribedTo->
            getUserName());
}
```

Listing 167: Test User Repository

```
/*******************************************************
* Function: testUserRepoFindMostSubscribedToUsersWithFileView()
*******************************************************
* Description: Tests that the returned list of most subscribed
* to users based on users who viewed the file is correct.
*******************************************************/
function testUserRepoFindMostSubscribedToUsersWithFileView(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $fileAidansApples->addFileUserViews($userNicole);
        $fileAidansApples->addFileUserViews($userCarol);
        $fileAidansApples->addFileUserViews($userLauren);

        $userNicole->addUserSubscribedTo($userAidan);
        $userNicole->addUserSubscribedTo($userLauren);
        $userNicole->addUserSubscribedTo($userCarol);

        $userLauren->addUserSubscribedTo($userAidan);
        $userLauren->addUserSubscribedTo($userCarol);

        $userLauren->addUserSubscribedTo($userAidan);

        $mostSubscribedToAfterViewingApples = $userRepo->
            findMostSubscribedToUsersWithFileView($fileAidansApples);
```

```
        $this->assertCount(3, count($mostSubscribedToAfterViewingApples
            ));
        $mostSubscribedTo = $mostSubscribedToAfterViewingApples->first
            ();
        $this->assertEquals($userAidan->getName(), $mostSubscribedTo->
            getUserName());
}
```

Listing 168: Test User Repository Find Most Subscribed To Users With File Dislike Function

```
/*******************************************************
* Function: testUserRepoFindMostSubscribedToUsersWithFileDisike()
*******************************************************
* Description: Tests that the returned list of most subscribed
* to users based on users who disliked the file is correct.
*******************************************************/
function testUserRepoFindMostSubscribedToUsersWithFileDisike(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $userRepo = $em->getRepository('Entity\\User');

        $fileAidansApples->addFileUserDislikes($userNicole);
        $fileAidansApples->addFileUserDislikes($userCarol);
        $fileAidansApples->addFileUserDislikes($userLauren);

        $userNicole->addUserSubscribedTo($userAidan);
        $userNicole->addUserSubscribedTo($userLauren);
        $userNicole->addUserSubscribedTo($userCarol);

        $userLauren->addUserSubscribedTo($userAidan);
        $userLauren->addUserSubscribedTo($userCarol);

        $userLauren->addUserSubscribedTo($userAidan);

        $mostSubscribedToAfterDislikingApples = $userRepo->
            findMostSubscribedToUsersWithFileDislike($fileAidansApples);

        $this->assertCount(3, count(
            $mostSubscribedToAfterDislikingApples));
        $mostSubscribedTo = $mostSubscribedToAfterDislikingApples->
            first();
        $this->assertEquals($userAidan->getName(), $mostSubscribedTo->
            getUserName());
}
```

12.1.4.5.2 File Repository Testing

Listing 169: Test File Repository Find One File By Id Function

```
/*******************************************************
* Function: testFileRepoFindOneFileById()
*******************************************************
* Description: Tests that the single file returned based
* on the id given is correct
*******************************************************/
function testFileRepoFindOneFileById(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $applesId = $fileAidansApples->getFileId();
        $findApplesById = $fileRepo->findOneFileById($applesId);
        $findApplesByIdActualId = $fileAidansApples->getFileId();
        $findApplesByIdName = $fileAidansApples->getFileName();

        $this->assertEquals($findApplesByActualId, $applesId);
        $this->assertEquals($findApplesByIdName, $fileAidansApples->
            getFileName());
}
```

Listing 170: Test File Repository Find One File By Name And Type Function

```
/*******************************************************
* Function: testFileRepoFindOneFileByNameAndType()
*******************************************************
* Description: Tests that the single file returned based
* on the name and type given is correct.
*******************************************************/
function testFileRepoFindOneFileByNameAndType(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $luggageType = $fileLaurensLuggage->getFileType();
        $luggageName = $fileLaurensLuggage->getFileName();
        $findByNameAndType = $fileRepo->findOneByFileNameAndType(
            $luggageName, $luggageType);

        $this->assertEquals($findByNameAndType->getFileName(),
            $luggageName);
```

```
        $this->assertEquals($findByNameAndType->getFileType(),
            $luggageType);
}
```

Listing 171: Test File Repository Find Most Relevant Files Function

```
/*******************************************************
* Function: testFileRepoFindMostRelevantFiles()
*******************************************************
* Description: Tests that the list of most relevant files
* returned is correct.
*******************************************************/
function testFileRepoFindMostRelevantFiles(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $fileSearchName = $fileAidansApples->getFileName();
        $fileSearchIsPaid = $fileAidansApples->getFileIsPaid();
        $fileSearchCategory = 'Comedy';
        $fileSearchType = $fileAidansApples->getFileType();

        $nameLikeSearchName = 'Applets';
        $fileDummyFile->setFileName($nameLikeSearchName);
        $fileDummyFile->setFileCategory($fileSearchCategory);
        $fileDummyFile->setFileType($fileSearchType);
        $fileDummyFile->setFileIsPaid($fileSearchIsPaid);

        $fileResults = $fileRepo->findMostRelevantFiles($fileSearchName
            , $fileSearchIsPaid, $fileSearchCategory, $fileSearchType);

        $this->assertCount(2,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $fileSearchName);
}
```

Listing 172: Test File Repository Find Most Liked Files Function

```
/*******************************************************
* Function: testFileRepoFindMostLikedFiles()
*******************************************************
* Description: Tests that the list of most liked files
```

```
* returned is correct
*****************************************************/
function testFileRepoFindMostLikedFiles(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $fileSearchName = $fileLaurensLuggage->getFileName();
        $fileSearchIsPaid = $fileLaurensLuggage->getFileIsPaid();
        $fileSearchCategory = 'Traveling';
        $fileSearchType = $fileLaurensLuggage->getFileType();

        $nameLikeSearchName = 'Lugging My Stuff Around';
        $fileDummyFile->setFileName($nameLikeSearchName);
        $fileDummyFile->setFileCategory($fileSearchCategory);
        $fileDummyFile->setFileType($fileSearchType);
        $fileDummyFile->setFileIsPaid($fileSearchIsPaid);
        $fileDummyFile->setFileLikes(10000);

        $fileResults = $fileRepo->findMostLikedFiles($fileSearchName,
            $fileSearchIsPaid, $fileSearchCategory, $fileSearchType);

        $this->assertCount(2,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $nameLikeSearchName);
}
```

Listing 173: Test File Repository Find Most Viewed Files Function

```
function testFileRepoFindMostViewedFiles(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $fileSearchName = $fileNicolesNews->getFileName();
        $fileSearchIsPaid = $fileNicolesNews->getFileIsPaid();
        $fileSearchCategory = 'Technology';
        $fileSearchType = $fileNicolesNews->getFileType();

        $nameLikeSearchName = 'NewsOn Technology';
        $fileDummyFile->setFileName($nameLikeSearchName);
        $fileDummyFile->setFileCategory($fileSearchCategory);
        $fileDummyFile->setFileType($fileSearchType);
        $fileDummyFile->setFileIsPaid($fileSearchIsPaid);
        $fileDummyFile->setFileViews(100000);
```

```
        $fileResults = $fileRepo->findMostViewedFiles($fileSearchName,
            $fileSearchIsPaid, $fileSearchCategory, $fileSearchType);

        $this->assertCount(2,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $nameLikeSearchName);
}
```

Listing 174: Test File Repository Find Newest Files Function

```
/*******************************************************
* Function: testFileRepoFindNewestFiles()
*******************************************************
* Description: Tests that the list of newest files returned
* is correct
*******************************************************/
function testFileRepoFindNewestFiles(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $fileSearchName = $fileAidansApples->getFileName();
        $fileSearchIsPaid = $fileAidansApples->getFileIsPaid();
        $fileSearchCategory = 'Comedy';
        $fileSearchType = $fileAidansApples->getFileType();

        $nameLikeSearchName = 'Applets';
        $fileDummyFile->setFileName($nameLikeSearchName);
        $fileDummyFile->setFileCategory($fileSearchCategory);
        $fileDummyFile->setFileType($fileSearchType);
        $fileDummyFile->setFileIsPaid($fileSearchIsPaid);
        $fileDummyFile->setFileDate(date("Y-m-d"));
        $fileResults = $fileRepo->findOldestFiles($fileSearchName,
            $fileSearchIsPaid, $fileSearchCategory, $fileSearchType);

        $this->assertCount(2 ,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $nameLikeSearchName);
}
```

Listing 175: Test File Repository Find Most Viewed Files After File Like Function

```
/*******************************************************
* Function: testFileRepoFindMostViewedFilesAfterFileLike()
*******************************************************
* Description: Tests that the list of most viewed files
* based on users who liked the file is correct
*******************************************************/
function testFileRepoFindMostViewedFilesAfterFileLike(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $fileDummyFile->addFileUserLikes($userAidan);
        $fileDummyFile->addFileUserLikes($userNicole);
        $fileDummyFile->addFileUserLikes($userLauren);
        $userAidan->addUserFileLikes($fileDummyFile);
        $userAidan->addUserFileViews($fileAidansApples);
        $userAidan->addUserFileViews($fileNicolesNews);
        $userAidan->addUserFileViews($fileLaurensLuggage);

        $userNicole->addUserFileLikes($fileDummyFile);
        $userNicole->addUserFileViews($fileAidansApples);
        $userNicole->addUserFileViews($fileNicolesNews);

        $userLauren->addUserFileLikes($fileDummyFile);
        $userLauren->addUserFileViews($fileAidansApples);

        $fileResults =$fileRepo->findMostViewedFilesAfterFileDislike(
            $fileDummyFile);

        $this->assertCount(3 ,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $fileAidansApples->getFileName());
}
```

Listing 176: Test File Repository Find Most Viewed Files After File Dislike Function

```
/*******************************************************
* Function: testFileRepoFindMostViewedFilesAfterFileDislike()
*******************************************************
* Description: Tests that the list of most viewed files based
* on users who disliked the file is correct.
*******************************************************/
function testFileRepoFindMostViewedFilesAfterFileDislike(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
```

```
        $fileRepo = $em->getRepository('Entity\\File');

        $fileDummyFile->addFileUserDislikes($userAidan);
        $fileDummyFile->addFileUserDislikes($userNicole);
        $fileDummyFile->addFileUserDislikes($userLauren);
        $userAidan->addUserFileDislikes($fileDummyFile);
        $userAidan->addUserFileViews($fileAidansApples);
        $userAidan->addUserFileViews($fileNicolesNews);
        $userAidan->addUserFileViews($fileLaurensLuggage);

        $userNicole->addUserFileDislikes($fileDummyFile);
        $userNicole->addUserFileViews($fileAidansApples);
        $userNicole->addUserFileViews($fileNicolesNews);

        $userLauren->addUserFileDislikes($fileDummyFile);
        $userLauren->addUserFileViews($fileAidansApples);

        $fileResults =$fileRepo->findMostViewedFilesAfterFileDislikes(
            $fileDummyFile);

        $this->assertCount(3 ,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $fileAidansApples->getFileName());
}
```

Listing 177: Test File Repository Find Most Liked Files After File View Function

```
/*******************************************************
* Function: testFileRepoFindMostLikedFilesAfterFileView()
*******************************************************
* Description: Tests that the list of most liked files
* based on users who viewed the file is correct.
*******************************************************/
function testFileRepoFindMostLikedFilesAfterFileView(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $fileDummyFile->addFileUserViews($userAidan);
        $fileDummyFile->addFileUserViews($userNicole);
        $fileDummyFile->addFileUserViews($userLauren);
        $userAidan->addUserFileViews($fileDummyFile);
        $userAidan->addUserFileLikes($fileAidansApples);
        $userAidan->addUserFileLikes($fileNicolesNews);
        $userAidan->addUserFileLikes($fileLaurensLuggage);
```

```
        $userNicole->addUserFileViews($fileDummyFile);
        $userNicole->addUserFileLikes($fileAidansApples);
        $userNicole->addUserFileLikes($fileNicolesNews);

        $userLauren->addUserFileViews($fileDummyFile);
        $userLauren->addUserFileLikes($fileAidansApples);

        $fileResults =$fileRepo->findMostLikedFilesAfterFileView(
            $fileDummyFile);

        $this->assertCount(3 ,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $fileAidansApples->getFileName());
}
```

Listing 178: Test File Repository Find Most Liked Files After File Like Function

```
/*******************************************************
* Function: testFileRepoFindMostLikedFilesAfterFileLike()
*******************************************************
* Description: Tests that the list of most liked files
* based on users who liked the file is correct
*******************************************************/
function testFileRepoFindMostLikedFilesAfterFileLike(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $fileDummyFile->addFileUserLikes($userAidan);
        $fileDummyFile->addFileUserLikes($userNicole);
        $fileDummyFile->addFileUserLikes($userLauren);
        $userAidan->addUserFileLikes($fileDummyFile);
        $userAidan->addUserFileLikes($fileAidansApples);
        $userAidan->addUserFileLikes($fileNicolesNews);
        $userAidan->addUserFileLikes($fileLaurensLuggage);

        $userNicole->addUserFileLikes($fileDummyFile);
        $userNicole->addUserFileLikes($fileAidansApples);
        $userNicole->addUserFileLikes($fileNicolesNews);

        $userLauren->addUserFileLikes($fileDummyFile);
        $userLauren->addUserFileLikes($fileAidansApples);
```

```
        $fileResults =$fileRepo->findMostLikedFilesAfterFileLike(
            $fileDummyFile);

        $this->assertCount(3 ,count($fileResults));
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(),
            $fileAidansApples->getFileName());
}
```

Listing 179: Test File Repository Find Most Liked Files After Subscribed To Function

```
/*********************************************************
* Function: testFileRepoFindMostLikedFilesAfterSubscribedTo()
*********************************************************
* Description: Tests that the like of most liked files
* by users who made a subscription to the user is correct.
*********************************************************/
function testFileRepoFindMostLikedFilesAfterSubscribedTo(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $userCarol->addUserSubscribers($userAidan);
        $userCarol->addUserSubscribers($userNicole);
        $userCarol->addUserSubscribers($userLauren);
        $userAidan->addUserSubscribedTo($userCarol);
        $userAidan->addUserSubscribedTo($userCarol);
        $userAidan->addUserSubscribedTo($userCarol);

        $userAidan->addUserFileLikes($fileDummyFile);
        $userAidan->addUserFileLikes($fileAidansApples);
        $userAidan->addUserFileLikes($fileNicolesNews);
        $userAidan->addUserFileLikes($fileLaurensLuggage);

        $userNicole->addUserFileLikes($fileDummyFile);
        $userNicole->addUserFileLikes($fileAidansApples);
        $userNicole->addUserFileLikes($fileNicolesNews);

        $userLauren->addUserFileLikes($fileDummyFile);
        $userLauren->addUserFileLikes($fileAidansApples);

        $fileResults =$fileRepo->findMostLikedFilesAfterSubscribedTo(
            $userCarol);

        $this->assertCount(4 ,count($fileResults));
```

```
        $firstResult = $fileResults->first();
        $this->assertEquals($firstResult->getFileName(), $fileDummyFile
            ->getFileName());
}
```

Listing 180: Test File Repository Find Most Liked User Files Function

```
/*******************************************************
* Function: testFileRepoFindMostLikedUserFiles()
*******************************************************
* Description: Tests that the list of most liked files
* by a user is correct.
*******************************************************/
function testFileRepoFindMostLikedUserFiles(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $userAidan->addUserOwnedFiles($fileDummyFile);
        $fileDummyFile->setFileOwner($userAidan);
        $fileDummyFile->setFileLikes(10000);
        $fileResults =$fileRepo->findMostLikedUserFiles();

        $this->assertCount(2, count($fileResults));
        $firstResult = $fileResult->first();
        $this->assertEquals($firstResult->getFileName(), $fileDummyFile
            ->getFileName());
}
```

Listing 181: Test File Repository Find Most Viewed User Files Function

```
function testFileRepoFindMostViewedUserFiles(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $userAidan->addUserOwnedFiles($fileDummyFile);
        $fileDummyFile->setFileOwner($userAidan);
        $fileDummyFile->setFileViews(10000);
        $fileResults =$fileRepo->findMostViewedUserFiles();

        $this->assertCount(2, count($fileResults));
        $firstResult = $fileResult->first();
        $this->assertEquals($firstResult->getFileName(), $fileDummyFile
            ->getFileName());
```

```
}
```

Listing 182: Test File Repository Find Newest User Files Function

```
/*******************************************************
* Function: testFileRepoFindNewestUserFiles()
*******************************************************
* Description: Tests that the list of the newest user files
* is correct
*******************************************************/
function testFileRepoFindNewestUserFiles(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $fileRepo = $em->getRepository('Entity\\File');

        $userAidan->addUserOwnedFiles($fileDummyFile);
        $fileDummyFile->setFileOwner($userAidan);
        $fileDummyFile->setFileUploadDate(date("Y-m-d"));
        $fileResults =$fileRepo->findMostLikedUserFiles();

        $this->assertCount(2, count($fileResults));
        $firstResult = $fileResult->first();
        $this->assertEquals($firstResult->getFileName(), $fileDummyFile
            ->getFileName());
}
```

12.1.4.5.3 Comment Repository Testing

Listing 183: Test Comment Repository Find One Comment By Id Function

```
/*******************************************************
* Function: testCommentRepoFindOneCommentById()
*******************************************************
* Description: Tests that the comment returned by the id
* given is correct
*******************************************************/
function testCommentRepoFindOneCommentById(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $commentRepo = $em->getRepository('Entity\\Comment');

        $findThisComment = $commentAidanOnAidan->getCommentId();
        $commentResult = $commentRepo->findOneCommentById();
```

```
        $this->assertEquals($commentResult->getCommentId(),
            $findThisComment);
}
```

Listing 184: Test Comment Repository Find Newest User Profile Comments Function

```
/**************************************************
* Function: testCommentRepoFindNewestUserProfileComments()
**************************************************
* Description: Tests that the list of newest user profile
* comments is correct.
**************************************************/
function testCommentRepoFindNewestUserProfileComments(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $commentRepo = $em->getRepository('Entity\\Comment');

        $userAidan->addUserProfileComments($commentAidanOnAidan);
        $userAidan->addUserProfileComments($commentNicoleOnAidan);
        $userAidan->addUserProfileComments($commentCarolOnAidan);

        $newestProfileComments = $commentRepo->
            findNewestUserProfileComments($userAidan);
        $this->assertCount(3,count($newestProfileComments));

        $firstResult = $newestProfileComments->first();
        $this->assertEquals($firstResult->getCommentId(),
            $commentCarolOnAidan->getCommentId());
}
```

Listing 185: Test Comment Repository Find Most Liked File Comments

```
/**************************************************
* Function: testCommentRepoFindMostLikedFileComments()
**************************************************
* Description: Tests that the list of most liked file
* comments is correct
**************************************************/
function testCommentRepoFindMostLikedFileComments(){
        $em = $this->get('hirevoice.neo4j.entity_manager');
        $commentRepo = $em->getRepository('Entity\\Comment');

        $fileAidansApples->addUserFileComments($commentNicoleOnApples);
```

```
        $fileAidansApples->addUserFileComments($commentCarolOnApples);

        $findMostLikedFileComments = $commentRepo->
            findMostLikedFileComments($fileAidansApples);

        $this->assertCount(2, count($findNewestFileComments));
        $firstResult = $findNewestFileComments->first();
        $this->assertEquals($firstResult->getCommentId(),
            $commentNicoleOnApples->getCommentId());
}
```